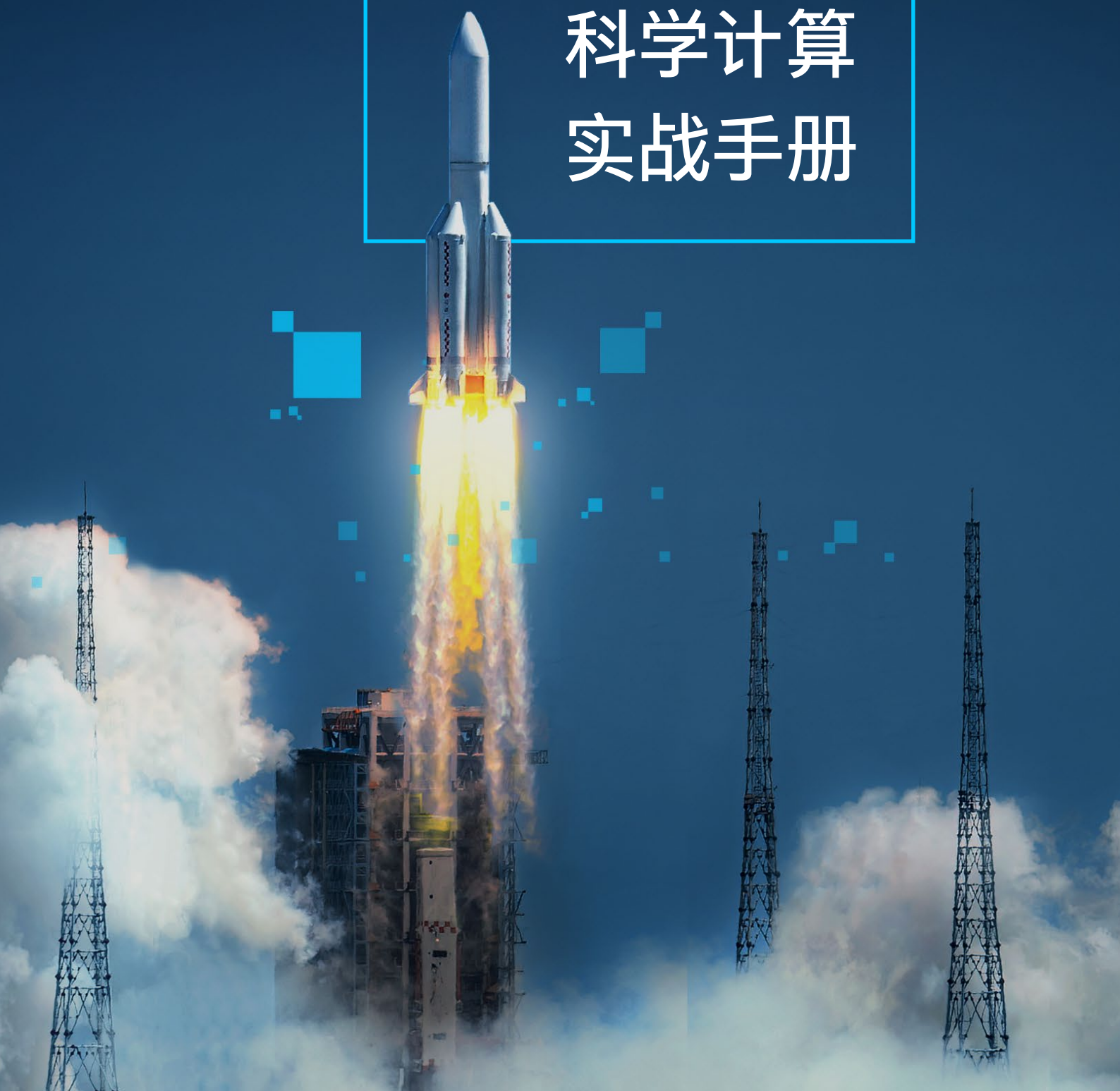
intel XEON

英特尔中国科学计算实战手册



Contents
目录

趋势篇

今天，在科学研究和技术实践各领域，以计算机和智能为代表的信息技术已成为加速创新的关键力量，作为现代科技三大支柱之一的科学计算，更是在其中发挥着无可替代的作用。尤其是一些领域，随着科学研究的深入，需要处理的数据量越来越大，算法也日益复杂，需要计算机系统大幅提高性能、加快处理速度来予以支撑。例如，在流体力学领域，一些仿真模拟场景中所需处理的网格动辄数以亿计；在气象预测领域，气象机构每年从卫星、飞机或观测站等获取的气象数据多达 PB 级别。在其它如天文、生命科学等领域同样如此，大规模方程计算和海量数据处理，虽然为探索未知开辟着新路径，但其计算过程就犹如黑洞一般会大幅榨取计算资源，让运算变得异常困难，也带来了巨大的成本。

在技术的落地实践、工程实现阶段，这一现象就更为突出。例如在制造行业，在设计研发阶段开展高精度的仿真模拟，不仅能大幅降低物理原型 / 实验的数量和成本，还能提高设计质量和效率，缩短新品研发上市时间，进而提升竞争力。但实施大规模仿真任务，往往需要数天乃至以周计才能完成，中间还可能还会因基础设施性能不足（例如内存带宽性能不足等）而中断，无法获得预期的效果。

为应对上述挑战，拥有更强性能的科学计算平台正应运而生且不断迭代进化，除了拥有比普通计算机系统更强的计算、存储和 IO 等基础能力，以及操作系统、驱动程序、文件系统、编译器和应用软件支持外，它还需要依托并行性（同时处理多项任务）和分布式（在多个节点处理任务）特性来实现更大规模的算力部署，以及计算的高效率和运行中的高稳定性。2023 年 6 月 25 日，英特尔宣布，Aurora 超级计算机在阿贡国家实验室完成部署，这成为全球首台峰值性能超过 2 Exaflops（1 Exaflops = 100 亿亿次浮点指令 / 秒）算力的超级计算机[1]。

图 1-1-1 Aurora 超级计算机

基于更强劲的性能和进一步优化的计算架构，科学计算平台，尤其是其中的佼佼者们正在千行百业中承担起越来越重要的角色。从传统的天文、物理、生物、气象等研究，到金融服务、生产制造、石油勘探等生产实践，再到新型产业如新材料研究、新药物研究、基因测序等工程探索，科学计算平台都正帮助人们加速科学发现、优化业务流程，进而创造更美好的生活。

- **制造领域**：由科学计算平台提供支持的计算机辅助工程（Computer Aided Engineering，CAE）正广泛用于产品设计与制造过程，例如在航空航天、船舶制造中，借助计算流体动力学（Computational Fluid Dynamics，CFD）和有限元分析（Finite Element Analysis，FEA）软件，能更好地模拟碰撞、噪声、振动、硬度和应力等，可加速结构分析，进而在降低研发制造成本的同时，为用户提供更优质的产品。
- **生命科学**：科学计算平台也被用于基因组分析、冷冻电镜数据分析等生命科学领域，助力健康医疗、制药产业的技术创新和应用。例如在制药行业，研究者可以借助科学计算平台与分子动力学模拟软件，来设计药物和模拟测试药物的有效性，而这不仅可缩短新药研发过程，也能够提高新药的安全性。
- **气象预测**：科学计算平台能通过对浩如烟海的气象数据展开处理和分析，来提升气象预测的精准度，进而帮助人们更有效应对灾害天气，如高温炎热、飓风等，也能对农业、风力发电等高度依赖气象预测领域的生产经营提供支持。

除此之外，科学计算平台同样也在太空探索、金融风险防范以及石油勘探等众多领域发挥巨大作用，此处不再一一赘述。而随着科学计算在千行百业重大创新的加速作用不断展现，其市场规模也得以高速扩展。有研究表明，2022 年全球科学计算市场规模已达 360 亿美元，预计到 2027 年这个数据将达 499 亿美元，年复合增长率（Compound Annual Growth Rate，CAGR）达 6.7%[2]。

随着科学计算应用范围的不断扩大，多样化的应用实践场景也对其平台性能提出了更高要求，需其通过架构创新、软件优化来应对更大规模计算带来的效率、成本等挑战。尤其随着人工智能（Artificial Intelligence，AI）、大数据（Big Data）以及云计算（Cloud Compute）等前沿 IT 技术不断被引入科学计算应用场景，使得在强化硬件基础设施之上，通过各类优化软件和加速库，来提升科学计算平台的效能，也成为了各行各业使用者所关注的焦点。

一直以来矢志于在科学计算领域发挥“核芯”作用的英特尔，除了提供英特尔® 至强® 可扩展处理器系列、英特尔® 至强® CPU Max 系列等先进算力平台产品外，也在持续通过英特尔® oneAPI、英特尔® AVX - 512 等软件和技术，为科学计算平台的优化和性能增强提供更多助力。在下一篇“应用优化实战篇”中，我们将就面向英特尔® 架构的 CAE 仿真分析、分子动力学以及生命科学相关软件的优化编译和运行，进行细节剖析和示例参考。

[1] 更多信息请参阅：https://www.alcf.anl.gov/aurora

[2] 数据援引自 marketsandmarkets 报告《High-performance Computing (HPC) Market by Compnent, Computation Type (Parallel Computing Distributed computing and Exascale Computing), Industry,Deployment,Server Price Band, Verticals & Region - 2027》：https://www.marketsandmarkets.com/Market-Reports/Quantum-High-Performance-Computing-Market-631.html


vPROJECT ANN Xv ADVANCED ARTIFICIAL NEURAL NETWORK FOR SPEACIAL PURPOSES

应用优化
实战篇

应用实践中的科学计算平台

今天，各类高性能的科学计算平台已在工业设计制造、生命科学、医疗健康、气象环境、航空航天获得了广泛运用，承担起产品设计、数据分析和模型优化等工作。与普通的计算场景不同，科学计算平台在实践应用中，对平台的基础算力、内存带宽、并行计算能力以及面向不同应用的专门加速库都有着较高要求。

因此在各个应用实践场景中，使用者在选择适合的硬件基础设施之外，也有必要针对场景的需求以及硬件基础设施的特性，对科学计算平台进行优化与增强。使用者部署在科学计算平台上的各类应用，如 OpenFOAM、Relion 等，通常是通过下载源代码（从 Github 网站或 Git 本地仓库）再进行编译的方式进行部署和运行，因此在编译和运行的过程中对各项参数进行合理配置，是帮助使用者们获得更优性能的有效手段（根据任务需求，通过各型英特尔® 编译器执行编译过程同样也是重要的优化途径）。

此外，几乎所有的科学计算应用都会采用多节点部署和并行计算的方式来提升计算效率，缩短处理时长，因此对多节点并行计算方式的优化也是使用者应当关注的重点。在实战中，使用者通常需要关注以下方面：

- **计算速度**：这一方面取决于所选择核心算力芯片的内核数量，主频、微架构设计以及末级缓存容量等，另一方面也应考虑芯片的指令集架构（Instruction Set Architecture，ISA），例如对 SIMD（Single Instruction Multiple Data，单指令多数据（Single Instruction Multiple Data，SIMD）的支持，这对于广泛使用并行计算的科学计算应用而言，无疑非常重要。同时，英特尔® 睿频加速技术（Intel® Turbo Boost Technology）、英特尔® 超线程技术（Intel® Hyper-Threading Technology）、增强型英特尔® SST（Enhanced Intel® Speed Select Technology）等基于英特尔® 架构的处理器性能增强，也能为科学计算任务的加速提供助力。

- **内存性能**：无论是流体动力学，还是基因组分析，科学计算应用所面临的一个共性问题是需要处理和传递的数据集和模型变得越来越大。例如在流体动力学中，计算任务面临的网格数量可能高达百亿。这一情况下，内存带宽性能也成为制约科学计算效率的重要因素。因此，除了选择支持 DDR4/DDR5 高性能内存产品的平台，引入高带宽内存（High Bandwidth Memory，HBM）也是一项重要选择。

- **并行计算**：执行并行计算以及多节点间的并行计算是提升科学计算效能的重要途径。使用者一方面可以借助非一致性内存访问（Non Uniform Memory Access，NUMA）等技术来实现多路并行算力优化；另一方面，英特尔® MPI 库等的引入，也可以使复杂的科学计算应用程序能够在基于英特尔® 架构的处理器及兼容相关架构的处理器的科学计算集群上运行得更好。

- **加速库**：科学计算任务中涉及大量或简单、或复杂的数学、物理以及其它计算，在编译时引入专门的加速库能够有效提升计算效率，例如快速傅立叶变换（Fastest Fourier Transform，FFT）计算在分子动力学，生命科学相关计算任务中有着非常频繁的使用。使用者可通过引入英特尔® oneAPI 数学内核库（英特尔® oneMKL）等加速库，不仅为科学计算任务提供基础线性代数子程序库（Basic Linear Algebra Subprograms，BLAS）来加速线性代数计算的效率，也为快速傅立叶变换等计算过程提供助力。

应用于 CAE 仿真分析的科学计算平台

CAE 仿真分析技术

随着科技的不断进步，各行业对于工程与产品设计的要求也精益求精。传统的设计方法已经不能满足全部需求，计算机辅助工程（CAE）的出现解决了这个问题。CAE 一般指用计算机系统对工程或产品进行各类分析，对其工作状态和运行模式进行模拟，及早发现设计缺陷，并验证功能和性能的可用性和可靠性。CAE 软件能帮助工程师对新产品、新系统进行设计、分析、优化和验证，以新的设计、验证方法来大幅提高工程与产品设计的效率和准确度。常见的 CAE 软件包括 ANSYS、ABAQUS 等。

通常而言，基于 CAE 软件开展的仿真分析可分为分析建模、前处理、求解计算、后处理等流程。常见的应用分类包括 FEA、CFD 等，其中：

- **有限元分析（FEA）软件**：可以帮助企业减少在产品或者流程的设计、优化或控制环节中原型测试的原型数量和测试次数。对于企业和研究机构来说，有限元仿真分析不仅仅可降低成本，更重要的是能够帮助企业或机构在激烈的市场竞争中增加优势，为研发投入带来更高的回报。

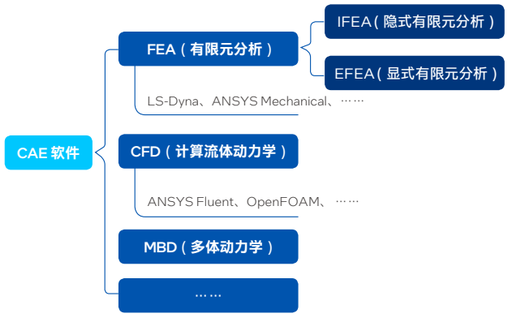


图 2-1-1 不同分类的 CAE 应用软件

- **计算流体动力学（CFD）软件**：CFD 是流体力学的一个分支，是在各种工程学科中对稳定或非稳定流体的流动进行建模的实践。它通过计算机模拟获得流体在特定条件下的数据，实现了用计算机系统代替试验装置来完成“计算试验”，为工程技术人员提供了实际工况模拟仿真的操作平台。CFD 工作负载往往涉及具有数百万乃至上亿个单元的复杂非结构化网格，对内存带宽性能更为敏感。

在面向 CAE 仿真分析的科学计算平台构建上有着丰富经验的英特尔，一直以来都通过其强劲的硬件产品性能和丰富全面的软件栈和加速库，为 CAE 仿真分析领域的软件提供强劲的支持和丰富的优化措施。下文中将就 OpenFOAM、Ansys Fluent 等常见 CAE 仿真分析软件在基于英特尔® 架构的平台上的优化编译和运行展开介绍。

面向仿真分析软件的英特尔优化

面向 OpenFOAM 的优化

作为一个 C++ 工具箱，OpenFOAM（Open source Field Operation And Manipulation）具有一系列面向 CFD 解决方案的自定义数据求解器（Solver）和前处理、后处理组件。其内置了一个独特的、高度可扩展的 CFD 软件开发工具套件（devkit），并包含了由这个 devkit 构建的一系列 CFD 应用程序。例如在一些版本中，使用者可方便地在其中描述偏微分方程的有限体积离散化，支持多面体网格，并支持大型并行计算等。这一系列的应用程序具有可便捷地进行定制与扩展的特点，方便了工业企业、学术研究和政府机构在大量相关领域中利用 OpenFOAM 开展高效的 CFD 仿真分析工作[3]。

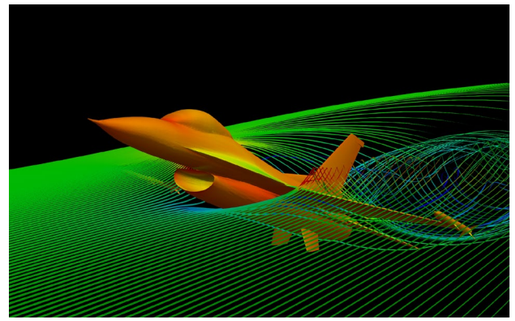


图 2-2-1 使用 OpenFOAM 开展高效的 CFD 仿真分析工作

OpenFOAM 的工作流程一般可分为问题定义与规划、创建计算网格、选择合理的模型、求解器与参数（例如边界条件、迭代次数等）以及对各类参数进行配置，启动求解器进行迭代计算以及后处理、结果分析等步骤。

[3] OpenFOAM 由 OpenFOAM 基金会根据 GNU 通用公共许可条款作为开源代码分发。OpenFOAM 由 OpenFOAM 创始人、CFD Direct 和 OpenFOAM 基金会董事 Henry Weller 领导的多个贡献者开发。OpenFOAM 的当前版本可在 https://openfoam.org/download/ 下载，可用于 Ubuntu Linux、其他 Linux 发行版以及该软件的 Windows 或 macOS 版本。用户也可以从源代码下载并编译 OpenFOAM。

在实践中，由于计算网格动辄百万起乃至上亿，OpenFOAM 的工作负载通常需要应对海量的计算任务，因此，OpenFOAM 通常会使用拆分算域法来执行并行计算，即，将计算域分解成一系列可单独执行的离散部分，并使用不同的处理器内核对其分别开展计算。同时，海量的计算网格也对计算平台的内存带宽等性能有着较高要求。基于上述工作特性，OpenFOAM 效能的发挥，一方面需要使用者为之提供有充分软硬件支持力（算力、内存带宽、开发生态等）的科学计算平台，另一方面也需要根据计算设施、应用场景的差别进行有针对性的优化。

■ 更强的科学计算平台支撑

由英特尔® 至强® 可扩展处理器系列等为核心构建的科学计算平台能帮助 OpenFOAM 的使用者获得更佳性能表现。以第四代英特尔® 至强® 可扩展处理器为例，全新微架构不仅每路配备多达 60 个内核，单核性能也比上一代产品更优。同时，该处理器也在以下方面对 OpenFOAM 工作负载提供助力：

- 提供了对 DDR5 内存的支持，提供的带宽和速度与上一代 DDR4 相比提高多达 1.5 倍，速率达到 4,800 MT/s；
- 具有多达 80 条 PCIe 5.0 通道，PCIe 5.0 的 I/O 带宽是上一代 PCIe 4.0 的两倍；
- 通过英特尔® 超级通道互联（Intel® Ultra Path Interconnect, 英特尔® UPI）2.0 提高多路带宽（高达 16 GT/s）；
- 提供 CXL 1.1 连接，在 CPU 和加速器之间创建统一且一致的内存空间，加速工作负载所需的数据吞吐。

值得一提的是，OpenFOAM 工作负载得益于全新的英特尔® 至强® CPU Max 系列对高带宽内存（HBM）的支持，性能上能够更上一层楼。作为一种采用 3D 堆叠技术的全新内存产品，HBM 能为 OpenFOAM 工作负载提供更高的内存带宽，大幅提升其数据吞吐性能。

在硬件平台之外，英特尔还为 OpenFOAM 工作负载提供了英特尔® oneAPI 工具套件这一基于新一代标准的英特尔® 软件开发工具，通过一个简化的、跨体系结构的编程模型来帮助使用者能简化相关开发、优化过程。同时，通过广泛的生态系统合作，英特尔一直致力于让 OpenFOAM 这样的开源应用在英特尔® 架构上有着更优表现，帮助各领域用户缩短项目时间，提升优化工作的效率。

■ 针对性的编译优化

由于特定的计算模式，OpenFOAM 工作负载受内存带宽影响更大。一些测量表明，目前大多数的 OpenFOAM 作业都不同程度受到了内存带宽限制的困扰。因此，在为面向 OpenFOAM 的科学计算平台提供第四代英特尔® 至强® 可扩展处理器、英特尔® 至强® CPU Max 等新一代处理器之外，使用者也可借助英特尔® 编译器，开展有针对性的编译优化。

在实战中，使用者可以参考以下代码示例来逐步开展编译优化：

 # 步骤 1

下载 OpenFOAM（本代码示例中使用了 OpenFOAM 4.1，可根据实际情况选择更新版本）及第三方软件（CGAL-4.8、boost_1_55_0、gmp-5.1.2、mpfr-3.1.2、metis-5.1.0，以上软件均可根据实际情况选择更新版本）

 # 步骤 2

矢量寄存器和单指令多数据（SIMD）硬件指令集是提升 CPU 并行处理效能的基本方法之一，因此为了更好利用算法 / 代码中的数据并行性，优化方案也加入了对英特尔® 高级矢量扩展 512（Intel® Advanced Vector Extensions 512, 英特尔® AVX - 512）的支持。英特尔® AVX - 512 指令集支持矢量级并行计算模式，能让 OpenFOAM 在运行时，每个内核同时使用两个矢量处理单元（其中每个单元能同时处理 16 个单精度（32 位）或 8 个双精度（64 位）的浮点数）。新一代的英特尔® 至强® 系列处理器都内置了英特尔® AVX - 512 技术，在使用时，需要在编译器选项中加入：-xCORE -AVX512。

在使用英特尔® 编译器构建 OpenFOAM 时，需要对相关参数进行设置（如 mpi），在面向 64 位系统进行编译时，可参考以下代码示例：

```
cd <YOUR_OPENFOAM_LOCATION>/OpenFOAM/OpenFOAM-4.1
1. vi etc/config.sh/mpi
  case "$WM_MPLIB" in - INTELMPI
  export MPI_ROOT=/opt/intel/compilers_and_libraries_2018.0.128/linux/mpi/intel64

2. vi etc/config.sh/settings
  a. case $WM_ARCH in Linux
    case `uname -m in > x86_64
      case $WM_ARCH_OPTION in
        i ) 32
        ii) 64
          > export WM_CC='icc'
          > export WM_CXX='icpc'
          > export WM_CFLAGS='-xCORE-AVX512 -O2 -no-prec-div -m64 -fPIC'
          > export WM_CXXFLAGS='-xCORE-AVX512 -O2 -no-prec-div -m64 -fPIC'

3. vi etc/bashrc
  > export FOAM_INST_DIR=/<YOUR_OPENFOAM_LOCATION>/OpenFOAM
  > export WM_COMPILER=Icc
  > export WM_MPLIB=INTELMPI
```


 # 步骤 3

创建安装文件，并在进行编译之前获取该文件的源代码，代码示例如下：

```
cd <YOUR_OPENFOAM_LOCATION>/OpenFOAM
vi setup
> source /opt/intel/compilers_and_libraries_2018.0.128/linux/bin/compilervars.sh intel64

> source /opt/intel/impi/2018.0.128/bin64/mpivars.sh
> source ./OpenFOAM-4.1/etc/bashrc
> cd ThirdParty-4.1

> sed -i -e 's/\(boost_version=\)boost-system/\1boost_1_55_0/' OpenFOAM-4.1/etc/config.sh/CGAL

> sed -i -e 's/\(cgal_version=\)cgal-system/\1CGAL-4.8/' OpenFOAM-4.1/etc/config.sh/CGAL

> cd ..
> source ./OpenFOAM-4.1/etc/bashrc WM_LABEL_SIZE=64 WM_COMPILER_TYPE=system FOAMY_HEX_MESH=yes
```

 # 步骤 4

构建第三方软件，包括使用英特尔® 编译器构建 gmp:

```
cd <YOUR_OPENFOAM_LOCATION>/OpenFOAM
source setup
cd ThirdParty-4.1/gmp-5.1.2

./configure --prefix=<YOUR_OPENFOAM_LOCATION>/OpenFOAM/ThirdParty-4.1/platforms/linux64/gmp-
5.1.2 CC=icc CXX=icpc CFLAGS='-xCORE-AVX512 CXXFLAGS='-xCORE-AVX512

mkdir platforms/linux64/gmp-5.1.2
make
make check
make install
```

使用英特尔® 编译器构建 mpfr:

```
cd <YOUR_OPENFOAM_LOCATION>/OpenFOAM
source setup
cd ThirdParty-4.1/mpfr-3.1.2

./configure --prefix=<YOUR_OPENFOAM_LOCATION>/OpenFOAM/ThirdParty-4.1/platforms/linux64/mpfr-
3.1.2 CC=icc CXX=icpc CFLAGS='-xCORE-AVX512 CXXFLAGS='-xCORE-AVX512 --with-gmp-
include=<YOUR_OPENFOAM_LOCATION>/OpenFOAM/ThirdParty-4.1/platforms/linux64/gmp-5.1.2/include --with-gmp-
lib=<YOUR_OPENFOAM_LOCATION>/OpenFOAM/ThirdParty-4.1/platforms/linux64/gmp-5.1.2/lib

mkdir platforms/linux64/mpfr-3.1.2
make
make check
make install
```

使用英特尔® 编译器构建 CGAL:

在构建 CGAL 的同时，使用者还需要构建 boost 库。CGAL 的 makefile 中调用 bootstap.sh 时，需要将 “bjam” 替换为 “b2”，并声明使用英特尔工具集构建库，同时添加英特尔® 编译器和优化标志 -xCORE -AVX512 的声明。以下是代码示例：

```
cd <YOUR_OPENFOAM_LOCATION>/OpenFOAM
source setup
cd ThirdParty-4.1
vi makeCGAL
> --with-toolset=intel-linux \
>&& ./b2 -j $WM_NCOMPPROCS install \
> ...
> ...
> -DCMAKE_C_COMPILER=icc \
> -DCMAKE_CXX_COMPILER=icpc \
> -DCMAKE_C_FLAGS='-xCORE-AVX512 \
> -DCMAKE_CXX_FLAGS='-xCORE-AVX512 \
```

为制作带有 gmp-5.1.2 和 mpfr-3.1.2 库的 CGAL，使用者还需要创建 gmp-5.1.2 和 mpfr-3.1.2 库，它们是在以下位置创建的：

```
cd Thirdparty-4.1.x/platforms/gmp-5.1.2/lib
cd Thirdparty-4.1.x/platforms/mpfr-5.1.2/lib
```

在编译 CGAL 时，库文件是在 lib64 文件夹下搜索的，因此使用者需要将 lib 文件夹重命名为 lib64，然后制作 CGAL 库。以下是代码示例：

```
cd <YOUR_OPENFOAM_LOCATION>/OpenFOAM
mv Thirdparty-4.1.x/platforms/gmp-5.1.2/lib Thirdparty-4.1.x/platforms/gmp-5.1.2/lib64
mv Thirdparty-4.1.x/platforms/mpfr-5.1.2/lib Thirdparty-4.1.x/platforms/mpfr-5.1.2/lib64
vi OpenFOAM-4.1.x/wmake/rules/General/CGAL
  > CGAL_INC = -I<YOUR_OPENFOAM_LOCATION>/OpenFOAM/ThirdParty-4.1/platforms/linux64/mpfr-
3.1.2/include\
-I<YOUR_OPENFOAM_LOCATION>/OpenFOAM/ThirdParty-4.1/platforms/linux64/gmp-5.1.2/include
  > CGAL_LIBS = -L<YOUR_OPENFOAM_LOCATION>OpenFOAM/platforms/linux64/mpfr-
3.1.2/lib$(WM_COMPILER_LIB_ARCH) \
-L<YOUR_OPENFOAM_LOCATION>/OpenFOAM/platforms/linux64/gmp-
5.1.2/lib$(WM_COMPILER_LIB_ARCH) \
source setup
cd Thirdparty-4.1.x
./makeCGAL CGAL-4.8 boost_1_55_0 gmp-5.1.2 mpfr-3.1.2
```

 # 步骤 5

使用英特尔® 编译器构建 Scotch 库，代码示例如下：

```
cd <YOUR_OPENFOAM_LOCATION>/OpenFOAM
cd Thirdparty-4.1.x/scotch_6.0.3/src/
rm Makefile.inc //删除之前的 similnk 到/Thirdparty-4.1.x/scotch_6.0.3/Makefile.inc
vi Make.inc/Makefile.inc.x86-64_pc_linux2.icc //对/Thirdparty-4.1.x/scotch_6.0.3/Make.inc/Makefile.inc.x86-64_pc_linux2.icc 进行更改
  > CCS = icc
  > CCP = mpiicc
  > CCD = mpiicc
  > CFALG = -xCORE-AVX512 -fPIC (additional flags)

ln –s Make.inc/Makefile.inc.x86-64_pc_linux2.icc Makefile.inc

make scotch
make prefix=$WM_THIRD_PARTY_DIR/platforms/linux64IccDPint64/scotch_6.0.3 libdir=$WM_THIRD_PARTY_DIR/platforms/linux64IccDPint64/lib install //在/Tirdparty-4.1.x/SCOTCH_6.03/include/SCOTCH.h 中将 SCOTCH_Num 从 int 更改为 int64_t

make ptscotch
make prefix=$WM_THIRD_PARTY_DIR/platforms/linux64IccDPint64/scotch_6.0.3 libdir=$WM_THIRD_PARTY_DIR/platforms/linux64IccDPint64/lib/intel64 includedir=$WM_THIRD_PARTY_DIR/platforms/linux64IccDPint64/scotch_6.0.3/include/intel64 install
```

 # 步骤 6

用英特尔® 编译器构建 metis-5.1.0 库

```
cd <YOUR_OPENFOAM_LOCATION>/OpenFOAM
cd Thirdparty-4.1.x/matis-5.1.0
vi Makefile
  > openmp = -qopenmp
  > cc = mpiicc

make config prefix=$WM_THIRD_PARTY_DIR/platforms/linux64IccDPint64/metis-5.1.0
make
make install
```

 # 步骤 7

使用英特尔® 编译器构建 OpenFOAM-4.1.x。使用者需要在 mplibINTELMPI 文件中提供正确的 IntelMPI 路径。将 include64 和 lib64 替换为 include 和 lib，代码示例如下：

```
cd <YOUR_OPENFOAM_LOCATION>/OpenFOAM/OpenFOAM-4.1.x
vi wmake/rules/linux64Icc/mplibINTELMPI
    >PINC = -isystem ${MPI_ARCH_PATH}/include
    >PLIBS = -L${MPI_ARCH_PATH}/lib -lmpi
cd ..
source setup
export WM_NCOMPPROCS=8
./Allwmake -j 8 2>&1 | tee output.log
```

 # 步骤 8

编译完成后，用户可以检查 output.log，并运行检查其是否正常工作，代码示例如下：

```
cd <YOUR_OPENFOAM_LOCATION>/OpenFOAM
source setup
cd OpenFOAM-4.1/tutorials/
./Allrun
```

Ansys Fluent

作为一款功能强大的计算流体动力学工具，Ansys Fluent 的应用范围涵盖了各种物理建模功能，可对工业应用中包括飞机机翼上的气流、熔炉燃烧、鼓泡塔、石油平台、血液流量、半导体制造、无尘室设计以及污水处理厂等具体应用在内的流动、湍流、热交换和各类反应进行建模。同时，也可以向使用者提供现代化用户友好型界面，在单窗口工作流程中简化从前处理到后处理的流程。

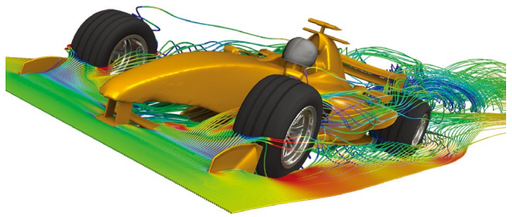


图 2-2-2 使用 Ansys Fluent 开展高效能的 CFD 仿真分析工作

轻松应对复杂工作任务的背后，是巨大的求解计算量，一些复杂的模拟可能需要数小时甚至数天才能完成。为了应对这一挑战，Ansys Fluent 需要引入科学计算平台来承载压力，而一直以来，Ansys Fluent 也以其高效的科学计算平台扩展性而著称，大规模分析可以让 Fluent 在多个计算处理芯片（包括 CPU、GPU 等）上轻松地求解。为了让 Ansys Fluent 工作负载获得更优的处理效能，其使用者也在不断寻求新的优化方案来提升其性能表现。

由英特尔提供的一系列硬件指令集和软件，包括英特尔® AVX-512、英特尔® oneMKL，能够对 Ansys Fluent 任务中的密集型计算任务提供优化，进而提升整体仿真分析的效率。这些硬件指令集和软件，在最新几代英特尔® 至强® 可扩展处理器平台上都获得了集成。用户可以在此基础上，结合英特尔® 架构处理器的多核优势对 Ansys Fluent 的求解器进行优化。

得益于全新的 HBM 内存以及其它集成技术，全新一代的英特尔® 至强® CPU Max 系列处理器对 Ansys Fluent 等需要大量内存参与的工作负载有着显著的性能加成。这一处理器平台的内核数量高达 56 个，并使用英特尔® 嵌入式多芯片互连桥接（Intel® Embedded Multi-Die Interconnect Bridge, 英特尔® EMIB）技术相互连接。

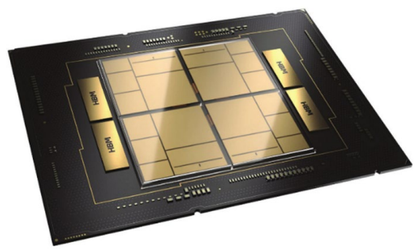


图 2-2-3 英特尔® 至强® CPU Max 系列处理器

该处理器的内存子系统中，包括了 64GB 内置 HBM2e 内存（为每个内核提供了超过 1GB 的 HBM 容量）、高达 112.5MB 的共享末级缓存和每路 8 个 DDR5-4800 内存通道。同时，内置的平台增强和硬件优化也有助于更大限度地提高 HBM 子系统的性能，包括：

- 重构的硬件预加载算法；
- 增强的非核心频率缩放技术；
- 面向本地内存请求的内核直连技术；
- 跨套接字一致性的增强型嗅探过滤器。

与 OpenFOAM 一样，Ansys Fluent 的工作流程大致可分为建立模型（例如涡流、粒子流模型）、定义参数特性、定义边界条件，启动求解器进行迭代计算以及后处理、结果分析等步骤。在 Ansys Fluent 工作任务中，会用到以下组件：

- **AMG 求解器：**Ansys Fluent 引入了代数多重网格（Algebraic multigrid，AMG）算法来驱动处理器的多个内核进行并行工作以提高效率。AMG 求解器可以使用一系列离散函数、模型、变量和方程来求解稀疏方程组。AMG 求解器在求解速度、稳健性和内存占用方面都非常高效；
- **平滑器（Smoother）：**一种可在多重网格级别之间转换，为整个方程组的中特定部分提供近似解，并显露重要特征的算法，是 AMG 求解器的关键组成部分。Ansys Fluent 原生的平滑器被称为 ILU 平滑器。

由于 ILU 平滑器具有间接内存访问、高内存带宽需求和循环携带依赖性等特性，其很难从矢量化计算中获得效率提升，为了解决这一问题，英特尔为 Fluent 提供了经优化的版本（使用者通过设定 platform=intel 选项来选择），从而使之能更大程度获得性能增强。同时，在结合英特尔® oneMKL 等技术后，用户可以将 ILU 平滑器优化为英特尔® oneMKL 稀疏 IDU 平滑器来有效提升 Ansys Fluent 的工作性能。

英特尔® oneMKL 是英特尔® oneAPI 基础工具包的一部分，其专门为加速解决大型计算问题所涉及的数学运算而构建，为稀疏或稠密线性代数提供了一系列高度优化的并行例程。除一直延续的性能优势外，英特尔® oneMKL 还专门添加了一组 SYCL 接口以实现在异构平台的运用。同时，其关键功能领域也包括了 BLAS 等线性代数例程、随机数生成器、矢量数学以及快速傅立叶变换（FFT）等，这些都可以对 Fluent 的关键性能实现有效增强。

新的基于英特尔® oneMKL 的稀疏 IDU 平滑器是基于英特尔® AVX-512 指令构建，能为 Fluent 提供高达 15% 的性能加速[4]，当出现 CFD 领域中常见的稀疏矩阵求解方程时，新的平滑器可通过算法为所涉及的部分方程组提供近似解。这一解决方案可为计算整体效率的提升提供助力。

在 Ansys Fluent 任务实战中，用户可以用单精度（single-precision）或双精度（double-precision）来执行英特尔® oneMKL 稀疏 IDU 平滑器，因此使用者需要先确定引入哪个库：

```
单精度: <ANSYS_HOME>/v201/fluent/lib/lnamd64/libmklsmoother_sp.so
多精度: <ANSYS_HOME>/v201/fluent/lib/lnamd64/libmklsmoother_dp.so
```

在上述每个接口库中，都包含作为成员的接口例程 “ilu”。使用者可按照以下代码示例对软件进行修改安装。

 # 步骤 1

在安装时，找到与 Ansys 产品绑定的英特尔® oneMKL 库：

```
<ANSYS_HOME>/v201/tp/IntelMKL
```

 # 步骤 2

在英特尔® MKL 下新建一个子目录以存储英特尔® oneMKL 库，例如将子目录命名为 2020.0.166：

```
cd <ANSYS_HOME>/v201/tp/IntelMKL
mkdir 2020.0.166
mkdir -parents 2020.0.166/linx64/lib/intel64/
```

 # 步骤 3

将英特尔® oneMKL 2020 黄金版库复制到 Fluent 安装目录中：

```
cp -p <MKL_HOME>/mkl/lib/intel64/*.so 2020.0.166/linx64/lib/intel64/
```

 # 步骤 4

找到 Fluent 可执行包装脚本，对引用的英特尔® oneMKL 版本进行更改：

```
<ANSYS_HOME>/v201/fluent/fluent20.1.0/bin/fluent
```

 # 步骤 5

修改脚本中的以下内容：

```
sys_prepend_ld_library_path
$FLUENT_INC/../tp/IntelMKL/2017.6.256/linx64/lib/intel64
```

修改至

```
sys_prepend_ld_library_path
$FLUENT_INC/../tp/IntelMKL/2020.0.166/linx64/lib/intel64
```

 # 步骤 6

最后在工作目录中，修改输入命令文件或 .jou 文件，用来调用 ILU 接口例程作为用户定义的平滑器。在执行求解步骤之前，将以下行添加到输入中：

```
(rpsetvar 'amg-coupled/user-defined-smoother ilu@libmklsmoother_sp.so )
```

另外，如果 libmklsmoother_sp.so 的副本已存于本地目录中，使用者可添加一个完整的路径来引用 Fluent 安装：

```
rpsetvar 'amg-coupled/user-defined-
smoother ilu@ANSYS_HOME/v201/fluent/lib/lnamd64/libmklsmoother _sp.so
```

（注：上述命令都引用了单精度（_sp.so）版本，使用者使用双精度（_dp.so））

[4]数据源自：Ansys, March 28, 2022. 2021 Annual Report. https://investors.ansys.com/static-files/d3e39ada-4e58-4dba-87f7-93e41eff55e4.

为了验证 Fluent 英特尔优化版本（-platform=Intel）、英特尔® oneMKL 稀疏 IDU 平滑器的加入两种优化对 Ansys Fluent 的性能提升，可以采用以下方案对优化方案进行测试，其中基准组采用了 Ansys Fluent 软件基础版本，测试组是使用英特尔® oneMKL 稀疏 IDU 平滑器等优化方案的不同版本。测试中分别对单节点和八节点硬件配置的平台进行了性能比较，测试中比较了以下四种软件版本：

- **软件版本 1**：使用 Fluent 平滑器的 Fluent 基础版本；
- **软件版本 2**：使用英特尔® oneMKL 稀疏 IDU 平滑器的 Fluent 基础版本；
- **软件版本 3**：使用 Fluent 平滑器的 Fluent 英特尔优化版本（-platform=Intel）；
- **软件版本 4**：使用英特尔® oneMKL 稀疏 IDU 平滑器的 Fluent 英特尔优化版本（-platform=Intel）。

从表 1 和表 2 所示的测试结果可以看出，无论是在单节点配置还是在八节点配置下，英特尔® oneMKL 稀疏 IDU 平滑器、面向英特尔架构优化的 Fluent 版本（或者两者协同）大部分情况下都能带来性能的提升。在单节点配置中，最大性能提升为 13%（combustor_12m 用例），而在八节点配置中，最大性能提升为 16%（combustor_12m 用例）。这些结果也说明，通常情况下英特尔® oneMKL 稀疏 IDU 平滑器、面向英特尔架构优化的 Fluent 版本两者带来的性能收益是互补的。

同时，测试在八节点配置上的成功，也证明英特尔® oneMKL 稀疏 IDU 平滑器、面向英特尔® 架构优化的 Fluent 版本带来的性能提升也能进行有效的扩展。

Single-Node: Intel Xeon Platinum 8280L Processor		Ansys Fluent Relative Solver Rating Normalized to Fluent Baseline Binaries with a Native Smoother			
Case	N Core Count	Fluent Baseline	Fluent Baseline Plus Intel MKL Sparse LDU Smoother	Fluent Optimized Binaries*	Fluent Optimized Binaries* Plus Intel MKL Sparse LDU Smoother
sedan_4m	56	1.00	1.04	1.01	1.04
aircraft_wing_14m	56	1.00	1.02	1.00	1.01
combustor_12m	56	1.00	1.05	1.08	1.13
pump_2m	56	1.00	1.11	1.02	1.11
rotor_3m	56	1.00	1.01	1.01	1.01
aircraft_wing_2m	56	1.00	1.00	1.00	0.99
exhaust_system_33m	56	1.00	1.05	1.00	1.05
landing_gear_15m	56	1.00	1.03	1.01	1.03

表 1 单节点配置性能对比（归一化）

Eight-Node Cluster: Intel Xeon Platinum 8280L Processor		Ansys Fluent Relative Solver Rating Normalized to Fluent Baseline Binaries with a Native Smoother			
Case	N Core Count	Fluent Baseline	Fluent Baseline Plus Intel MKL Sparse LDU Smoother	Fluent Optimized Binaries*	Fluent Optimized Binaries* Plus Intel MKL Sparse LDU Smoother
sedan_4m	448	1.00	0.97	1.03	1.00
aircraft_wing_14m	448	1.00	1.01	1.00	0.98
combustor_12m	448	1.00	1.04	1.12	1.16
pump_2m	448	1.00	1.00	1.03	1.01
rotor_3m	448	1.00	0.98	1.03	0.99
aircraft_wing_2m	448	1.00	0.96	1.01	0.93
exhaust_system_33m	448	1.00	1.05	1.01	1.05
landing_gear_15m	448	1.00	1.02	1.02	1.02
combustor_71m	448	1.00	1.13	1.00	1.13
f1_racecar_140m	448	1.00	1.15	1.01	1.15
open_racecar_280m	448	1.00	1.08	1.01	1.09

表 2 八节点配置性能对比（归一化）

为验证 HBM 内存（基于英特尔® 至强® CPU Max 系列处理器）对 Ansys Fluent 的优化效果，英特尔也设计了面向三种不同工作负载的测试，并在英特尔® 至强® CPU Max 系列、第四代英特尔® 至强® 可扩展处理器以及第三代英特尔® 至强® 可扩展处理器三种处理器平台上进行了对比测试。测试中的工作负载包括：

- F1_Racecar_140M：一辆方程式赛车周围的空气动力学流动模拟，1.4 亿个单元；
- Open_Racecar_280M：开放式车轮赛车周围的外部空气动力学流动模拟，2.8 亿个单元；
- Combustor_71m：流经燃烧器的燃烧模拟，7,100 万个单元。

如图 2-2-4 所示，通过比较，英特尔® 至强® 处理器 Max 系列集成的 HBM 的应用优势显而易见。此外，英特尔® 至强® CPU Max 系列与第四代英特尔® 至强® 可扩展处理器的性能也显著优于前代第三代英特尔® 至强® 可扩展处理器。

图 2-2-4 Ansys Fluent 在三种工作负载下，基于不同英特尔® 架构处理器平台的性能对比

应用于分子动力学的科学计算平台

分子动力学技术

分子动力学（MD）是在经典力学（例如牛顿力学）等的框架下，从给定的初始条件和参数出发，对分子的微观结构机制、动态轨迹行为等进行模拟，从而获得相关数据的有效方法。这种模拟既适合于对蛋白、核酸、多糖等生物大分子的研究，也可应用于材料结构、溶液小粒子等结构体系中。由于分子动力学可兼顾更大空间分辨率上的微观结构信息，以及更精细时间分辨率上的结构动力学信息，因此其在生物、材料等研究场景中都可发挥巨大作用，深受相关领域研究者的重视，具有广泛的应用前景。

由于这种微观级别的模拟很难通过人工计算的方式来完成，因此分子动力学从一开始就是一门与计算机科学有着紧密联系的学科，需要在性能强大的科学计算平台上展开。一般而言，分子动力学模拟的主要步骤包括确定起始构型、选用适当力场和模拟软件、构建体系和能量最小化、平衡过程、数据采集以及数据分析。如今，面向不同领域，已经出现了丰富的模拟软件包，例如 LAMMPS、NAMD、VASP 和 CP2K 等供研究者使用。

在选择合适的模拟软件包的同时，为分子动力学模拟过程选择合适的硬件基础设施，并根据模拟场景与软件包的特性进行专门的优化也是必不可少的步骤。在面向分子动力学的科学计算平台构建上有着丰富经验的英特尔，一直以来都通过其不断迭代更新的处理器平台和丰富全面的软件栈和加速库，为分子动力学模拟领域的软件包提供越来越强的算力支持和专门的性能优化。下文中将就 LAMMPS、NAMD、VASP 以及 CP2K 等常见分子动力学模拟软件包在基于英特尔® 架构的平台上的优化编译和运行展开介绍。

面向英特尔® 架构的平台的分子动力学软件优化

面向英特尔® 架构平台的 LAMMPS 优化编译与运行

作为经典的用于分子动力学模拟的开源软件包，LAMMPS（Large-scale Atomic/Molecular Massively Parallel Simulator）不仅支持多种粒子类型，如原子、聚合物分子、生物分子、金属、颗粒等的模拟，也支持多种力场，并可通过 MPI 和空间区域分解等方式支持并行模拟。与其它分子动力学软件相比，LAMMPS 可以对分子动力学中的单元粒子、相互作用和积分器等进行抽象并提供可灵活配置的 API。基于此，LAMMPS 实现了对更多粒子类型和力场的支持，使模拟对象更为广泛而不再限于某一门类，例如其在材料体系的模拟中具有非常广泛的应用，同时也对生物分子的模拟有着良好的支持。更多详细信息，可参阅 LAMMPS 官方文档[5]。

LAMMPS 的模拟过程无疑需要大量算力予以支持。除了使用并行模式以外，为 LAMMPS 提供强劲算力的平台也必不可少，而英特尔® 至强® 可扩展处理器系列一贯的高水准性能表现，堪称 LAMMPS 为分子动力学研究提供高水平输出的最佳拍档之一。

为了让基于英特尔® 架构处理器的 LAMMPS 在运行时，在保证精准结果的同时具有更优的计算效能，英特尔根据自身产品特性，为 LAMMPS 提供了可在大多数场景下获得更佳性能的一系列软硬件部署、配置方法与建议。本文的方法与建议是基于第四代英特尔® 至强® 可扩展处理器以及英特尔® 至强® CPU Max 系列等平台。

第四代英特尔® 至强® 可扩展处理器基于平衡、高效的架构构建，该架构可有效提升内核性能、内存和 I/O 带宽，为处理从数据中心到边缘的各种工作负载提速。其针对多样化的工作负载类型（例如 AI）和性能需求进行了优化，并通过平衡的架构以及多种集成加速和先进的安全功能来帮助用户将迫切的工作负载安全地放置在从边缘到云的最佳性能位置上。其中，它增强的基础性能、更多英特尔® UPI 以及英特尔® AVX - 512 将都可为 LAMMPS 的性能提升提供更为显著的助力。

■ 编译前设置

在第四代英特尔® 至强® 可扩展处理器平台上使用 LAMMPS 之前，可进行以下硬件设置。

在 BIOS 中启用（Enabled）以下三项设置：

- **英特尔® 睿频加速技术**：这一技术允许处理器在低于当前功率或温度规格的情况下自动提高频率。
- **英特尔® 超线程技术**：这一技术允许多线程应用在每个处理器中核内并行执行两个线程，即线程可运行在两倍于物理核的逻辑核上。
- **SNC（Sub-Numa Cluster）**：通过改进远程处理器访问，这一技术改进了之前处理器提供的片上集群（COD）选项。同时在操作系统级别，启用 SNC 的双处理器服务器将显示 8 个 NUMA 域。其中 4 个域在同一个处理器上，另外 4 个可通过 UPI 到达远程处理器。因此启用 SNC 将获得更好的性能表现。

[5] LAMMPS 官网：https://www.lammps.org/

内存配置 / 设置：

- 一般而言，由于模拟系统会扩展到多个节点，因此每个节点的内存空间可配置得小一点。

存储 / 磁盘配置 / 设置：

- 强烈建议使用固态盘（SSD）作为操作系统和安装 LAMMPS 的主驱动器以避免 I/O 瓶颈。也建议使用大容量固态盘来存储已有项目，尤其当使用 VMD 软件来创建模拟过程的视频、动画或电影。

网络配置 / 设置：

- 当使用英特尔® MPI 库（科学计算平台中常用的并行化库）在多个节点上并行处理数据集时，更易获得 LAMMPS 的最佳性能。

同时，为了让基于英特尔® 架构处理器的 LAMMPS 在运行时获得更佳性能，其本身也需要进行一些配置优化。LAMMPS 对各类型的仿真模型都有着很好的支持。如图 2-3-1 所示，这是其在长距离静电分子系统（Molecular systems with long-range electrostatics）中的工作流程。可通过以下设置来提升 LAMMPS 性能：

图 2-3-1 LAMMPS 在长距离静电分子系统中工作流程

步骤 3、步骤 6（可选）以及步骤 9 并非每个轮次都需要运行；

步骤 7 可通过 newton off 设置来关闭；

步骤 6 也可以通过长线程（Long-Range Thread，LRT）模式设置在单独的超线程上运行；

步骤 4 与步骤 5 可以采用并行计算模式

■ 代码编译

在完成优化配置后，使用者可以在面向英特尔® 架构处理器的上优化编译 LAMMPS。LAMMPS 的官方源代码包中内置了英特尔® 软件包（Intel® Package），但该包在编译时须与其它包体一起安装。英特尔® 软件包可以加速 LAMMPS 在英特尔® 架构处理器上的仿真速度。使用者可按照以下步骤和代码示例进行编译。

 # 步骤 1

从 Github（Git repository：http://github.com/lammps/lammps）下载 LAMMPS 并安装英特尔® 软件包：

```
git clone -b stable https://github.com/lammps/lammps.git lammps
```

```
cd lammps/src
make yes-intel
```

 # 步骤 2

使用英特尔® oneAPI 工具套件进行编译，编译后将得到 lmp_intel_cpu_intelmpi 二进制文件：（编译器和链接器设置在以下文件中：src/MAKE/OPTIONS/Makefile.intel_cpu_intelmpi）

```
source /opt/intel/oneapi/setvars.sh
make intel_cpu_intelmpi -j
```

 # 步骤 3

执行 LAMMPS，加入英特尔® 软件包优化的最简方法是在 LAMMPS 命令行中添加 -sf Intel 开关。这可令优化设置自动加载到仿真过程中。所使用的 OpenMP 线程数可以通过 OMP_NUM_threads 环境变量控制，也可通过：-pk intel 0 omp $N 来为 N 个 OpenMP 线程添加。

```
mpirun -np 72 -ppn 36 lmp_intel_cpu_intelmpi -sf intel -in in.script
# 2 nodes, 36 MPI tasks/node, $OMP_NUM_THREADS OpenMP Threads
mpirun -np 72 -ppn 36 lmp_intel_cpu_intelmpi -sf intel -in in.script -pk intel 0 omp 2 mode double
# Use 2 OpenMP threads for each MPI task, use double precision
```

■ 性能调整

- 当处理器的每个物理内核都运行一个 MPI 并行任务时，LAMMPS 性能表现良好。当英特尔® 超线程技术开启时，每个内核最适合运行两个 OpenMP 线程。
- 将 Newton 设置更改为 关闭（off）可提高简单双体电位（如 lj/cut）的“与 / 或”可扩展性。当在内置英特尔® AVX - 512 的处理器上使用 LRT 模式时，它也可进一步助力性能提升。
- LRT 模式是英特尔软件包中的一个选项，当在支持英特尔® 超线程技术的处理器上使用 PPPM（Particle - Particle-Particle - Mesh，质点 - 质点 - 质点 - 网格）方法进行长距离静电处理时，可有效提高性能。其可为每个 MPI 任务都生成一个额外的线程，专门用于执行一些 PPPM 计算和 MPI 通信。此功能要求 LAMMPS 在编译时，在 makefile 中设置预处理器标志 -DLMPINTELUSELRT（makefile.intelcpuintelmpi 的默认值）。
  - 使用 LRT 时，设置环境变量 KMP_AFFINITY=none。
  - 启用 LRT 模式，需要指定 OpenMP 线程的数量比正常运行时的线程数量少一个。然后添加 lrt-yes 选项：

```
Running without LRT mode:   -pk intel 0 omp 4
Running with LRT mode:      -pk intel 0 omp 3 lrt yes
```


■ 运行标准 LAMMPS 基准测试

LAMMPS 内置了涵盖各种不同仿真模型的基准测试脚本。以下步骤将运行以下基准测试：原子流体、蛋白质、嵌入原子法的铜、耗散粒子动力学、AIREBO 力场的聚乙烯、具有三体 Tersoff 模型的硅、具有三体 Stilling-Weber 势的硅、使用三体势的粗粒水以及液晶仿真。

运行基准测试前，必须在 LAMMPS 编译前安装以下软件包：

```
make yes-asphere yes-class2 yes-dpd-basic yes-kspace yes-manybody yes-misc yes-molecule yes-mpiio yes-opt yes-replica yes-rigid yes-intel
```

切换到基准测试所在目录：

```
cd lammps/src/INTEL/TEST
```

将 PCORES 设置为系统中物理内核的数量，并运行基准测试（最后打印来自报告时间步长 / 秒的日志文件的摘要性能数字，越高越好）：

```
PCORES=`lscpu | awk '$1=="Core(s):"{t=NF; cores=$t}$1=="Socket(s):"{t=NF; sockets=$t}END{print cores*sockets}'`;
sed -i "s/36/$PCORES/g" run_benchmarks.sh; sed -i 's/"2"/"12"/g' run_benchmarks.sh
./run_benchmarks.sh
```

■ 顺序进行下载、编译和基准测试

在一些安装了标准英特尔® oneAPI 工具套件的系统中，以下命令可使系统按步骤进行 LAMMPS 的下载和编译，并对其进行基准测试（某些配置可能需要修改）：

```
source /opt/intel/oneapi/setvars.sh; git clone -b stable https://github.com/lammps/lammps.git lammps; cd lammps/src; make yes-asphere yes-class2 yes-dpd-basic yes-kspace yes-manybody yes-misc yes-molecule yes-mpiio yes-opt yes-replica yes-rigid yes-openmp yes-intel; make intel_cpu_intelmpi -j; cd INTEL/TEST; PCORES=`lscpu | awk '$1=="Core(s):"{t=NF; cores=$t}$1=="Socket(s):"{t=NF; sockets=$t}END{print cores*sockets}'`; sed -i "s/36/$PCORES/g" run_benchmarks.sh; sed -i 's/"2"/"12"/g' run_benchmarks.sh; ./run_benchmarks.sh
```

面向英特尔® 架构平台的 NAMD 优化编译与运行

NAMD（NAnoscale Molecular Dynamics）是一种旨在对大型生物分子系统开展高性能模拟的并行分子动力学应用软件。借助 Charm++ 并行对象，NAMD 在典型的仿真环境中可扩展到数百个核心（core），在最大化仿真环境中可扩展到超过 500,000 个核心。NAMD 使用流行的分子图形程序 VMD 进行模拟设置和轨迹分析，也可与 AMBER、CHARMM 和 X-PLOR 文件相兼容。

对并行计算有着更优支持的英特尔® AVX - 512 无疑是 NAMD 提升模拟效能的良好选择。基于英特尔® AVX - 512 扩展的 NAMD 优化能显著提高其在英特尔® 架构处理器平台上的性能表现。从 2.15 版本之后，NAMD 就包含了该优化项，使用者也可以从 Git Repository 中获取最新的源代码并编译使用。只要模拟过程支持 SIMD 方式，这些编译过程都将默认使用英特尔® AVX - 512 优化。在 NAMD 运行初始化时，其将显示以下信息表示已基于英特尔® AVX - 512 的优化：

```
MIXED PRECISION AVX-512 TILES OPTIMIZATIONS: ENABLED
```

使用者如需取消优化项，可在命令行中添加 +notiles 选项，或在输入模拟脚本中添加 useAVXTiles no。

一般而言，上述编译、运行的步骤与典型的 NAMD 运行方法并无二致。但建议使用者选择最新的 NAMD 源代码，同时选择合适的 NAMD “结构” 文件，以便选择所需的编辑器标记进行优化也是必要的。

■ 编译 NAMD 准备工作

在编译经优化的 NAMD 之前，需要进行准备工作，包括下载源代码与基准测试脚本等。其中，Charm++ 是 NAMD 必需的组件，而工具命令语言（Tool Command Language，TCL）库是一个可选组件，可用于完全的脚本支持。TCL 是一种简易易扩展的脚本语言，可用于在各类交互式程序中发布命令，完成自动化批处理工作。

 # 步骤 1（可选项）

下载 TCL。如果不需要 TCL 支持，或系统上已经安装了 TCL（在某些系统上，可使用 locate libtcl8.5 检查现有 TCL），则可跳过此步骤。

```
$ git clone https://github.com/tcltk/tcl.git -b core-8-5-branch --depth 1
$ NAMD_TCL_BASE=`pwd`/tcl
```

 # 步骤 2 下载 Charm++ 和 NAMD：

```
$ git clone https://charm.cs.illinois.edu/gerrit/namd.git -b release-2-15-alpha-1 --depth 1
$ cd namd; git clone https://github.com/UIUC-PPL/charm.git -b v6.10.2 --depth 1
$ cd ..
```

在基础测试选择上，使用者可借助 NAMD 网站上流行的 APOA1 和 STMV 基准来进行。这些基准测试的模拟参数，可根据对性能影响的程度来进行设置。一般地，可使用标准模拟参数，并根据 NAMD 开发者的基准测试建议进行设置，在增加模拟持续时间的同时减少能量输出。以下是下载基准测试脚本并执行设置的方法。

 # 步骤 1

为创建一个相关工作目录。

```
$ mkdir namd_global
$ cd namd_global
```

 # 步骤 2

从 NAMD 网站下载基准测试脚本并对其进行设置，在实现更长时间运行的同时减少能量输出。

```
$ wget http://www.ks.uiuc.edu/Research/namd/utilities/apoa1.tar.gz
$ wget http://www.ks.uiuc.edu/Research/namd/utilities/stmv.tar.gz
$ tar -zxvf apoa1.tar.gz; tar -zxvf stmv.tar.gz
$ sed -i -e '/numsteps/s/500/1000/' apoa1/apoa1.namd stmv/stmv.namd
$ sed -i -e "/outputtiming/a\outputenergies 600" apoa1/apoa1.namd
$ sed -i -e '/outputEnergies/s/20/600/' stmv/stmv.namd
```

■ 基于单进程基准测试编译 NAMD 并执行基准测试

单进程可执行文件模式的编译和运行过程较为简单，但其运行仅限于单个节点和单个 NAMD 通信线程。当每个核心的原子数较少时，即使在单个节点上运行，具有多个通信线程的多进程模式也可提高性能。下文描述了以上两种模式的编译过程，首先是单进程的 NAMD 可执行文件编译及测试过程：

 # 步骤 1

设置编译器和库环境。在这里，我们使用英特尔® MPI 库、英特尔® oneMKL 以及英特尔® 编译器。使用英特尔® oneAPI，只需一个命令就可以为 Bash shell 设置环境。

```
$ source /opt/intel/oneapi/setvars.sh
```

 # 步骤 2

编译 TCL 库（可选）

```
$ cd tcl/unix; ./configure --disable-shared --prefix=$NAMD_TCL_BASE
$ make install -j; cd ../../
```

 # 步骤 3

编译单进程 Charm++ 库

```
$ export CC=icc; export CXX=icpc; export F90=ifort; export F77=ifort
$ cd ./namd/charm
$ ./build charm++ multicore-linux64 iccstatic --with-production "-O3 -ip -xCORE-AVX512 -qopt-zmm-usage=high"
$ cd ../
```

 # 步骤 4

编译单进程 NAMD 可执行文件。如在不使用 TCL 的情况下编译，需要将 --TCL prefix$NAMD_TCL_BASE 相应更改为 --without TCL。

```
$ ./config Linux-AVX512-icc --with-mkl --tcl-prefix $NAMD_TCL_BASE
$ cd Linux-AVX512-icc; make -j; cd ../../
```

 # 步骤 5

运行基准测试。对于工作负载较小或节点数较高的场景，可以在不使用所有超线程的情况下获得更好的性能。NAMD 性能可通过以纳秒 / 天（ns/ 天）为单位的模拟速率来衡量，数值越高越好。这里，使用者可选择使用英特尔® oneAPI 工具套件中的英特尔® TBB（Threading Building Blocks）内存分配技术来实现性能的小幅度改善。

```
$ export LD_PRELOAD="`ls $TBBROOT/lib/intel64/gcc*/libtbbmalloc_proxy.so` $LD_PRELOAD"
```

 # 步骤 6

通过命令行，设置处理节点上用于 NAMD 运行的内核总数，以及提取性能的变量。

```
$ N_CORES=`grep processor /proc/cpuinfo | wc -l`
$ H_CORES=`expr $N_CORES / 2`
$ GET_PERF="\$2==\"Benchmark\"{n++; s+=log(\$8); print \"Instant:\",1/\$8,\" ns/day\"}END{print \"Final:\",1/exp(s/n)}"
```

 # 步骤 7

运行 APOA1 和 STMV 基准测试。

```
$ ./namd/Linux-AVX512-icc/namd2 +p $N_CORES +setcpuaffinity ./apoa1/apoa1.namd | awk "$GET_PERF"
$ ./namd/Linux-AVX512-icc/namd2 +p $H_CORES +setcpuaffinity ./apoa1/apoa1.namd | awk "$GET_PERF"
$ ./namd/Linux-AVX512-icc/namd2 +p $N_CORES +setcpuaffinity ./stmv/stmv.namd | awk "GET_PERF"
```

■ 基于多进程基准测试编译 NAMD 并执行基准测试

对于多节点的场景，最优的编译和运行选项很大程度上取决于集群的配置。使用者可使用 Charm++ 的 MPI 后端（也可以使用 OFI），并使用英特尔® MPI 启动命令。

 # 步骤 1

基于英特尔® MPI 库构建 Charm++。

```
$ CC=icc; CXX=icpc; F90=ifort; F77=ifort; MPICXX=mpiicpc; MPI_CXX=mpiicpc
$ I_MPI_CC=icc;I_MPI_CXX=icpc;I_MPI_F90=ifort;I_MPI_F77=ifort
$ export I_MPI_CC I_MPI_CXX I_MPI_F90 I_MPI_F77 CC CXX F90 F77 MPICXX MPI_CXX
$ cd ./namd/charm
$ ./build charm++ mpi-linux-x86_64 smp mpicxx --with-production "-O3 -ip -xCORE-AVX512" -DCMK_OPTIMIZE -DMPICH_IGNORE_CXX_SEEK
$ cd ../
```

 # 步骤 2

基于英特尔® MPI 库编译 NAMD。如果编译时不使用 TCL，需要将 --TCL prefix$NAMD_TCL_BASE 更改为 --without TCL。

```
$ cp arch/Linux-AVX512-icc.arch arch/Linux-AVX512MPI.arch
$ ./config Linux-AVX512MPI --charm-base ./charm --charm-arch mpi-linux-x86_64-smp-mpicxx --with-mkl --tcl-prefix $NAMD_TCL_BASE
$ cd Linux-AVX512MPI
$ make -j
$ cd ../../
```

 # 步骤 3（可选步骤）

使用英特尔® oneAPI 工具套件中的英特尔® TBB 内存分配。

```
$ export LD_PRELOAD="`ls $TBBROOT/lib/intel64/gcc*/libtbbmalloc_proxy.so` $LD_PRELOAD"
```

 # 步骤 4

设定每个节点使用的物理内核数量。

```
$ N_CORES=`grep processor /proc/cpuinfo | wc -l`
```

步骤 5（可选步骤）

当每个物理内核分配到的原子数量较小时，调整物理内核数量以避免启用英特尔® 超线程技术。

```
$ N_CORES=`expr $N_CORES / 2`
```


 # 步骤 6

选择在每个节点上使用的 NAMD 进程数量。最佳数量取决于每个物理内核的原子数和系统配置。选择该数字时应确保 N_CORES 设置是可整除的，如以下代码示例中使用 4。

```
$ NPPN=4
```

 # 步骤 7

设置 MPI 进程总量以及与 NAMD 通信和工作线程的关联标志（假设 $NODES 已设置为节点数量）

```
$ NMPI=`expr $NPPN \* $NODES`
$ NAFFIN=`echo $N_CORES $NPPN | awk '{p=($1-$2)/$2; c=$1-1; f=p+1; print "+ppn",p,"+commap ","0-"c":"f,"+pemap 1-"c":"f","p}'`
```

 # 步骤 8

执行程序。

```
$ mpirun -ppn $NPPN -f $HOSTFILE -np $NMPI ./namd/Linux-AVX512MPI/namd2 $NAFFIN ./stmv/stmv.namd
```

使用英特尔® oneAPI Base 工具套件和 HPC 工具套件编译 VASP

VASP（Vienna Ab-Initio Simulation Package）是由维也纳大学 Hafner 小组开发的一个原子尺度的材料模拟软件包，其核心工作方法是基于赝势方法和平面波基组进行第一性原理（Ab-Initio）量子力学 - 分子动力学的模拟。VAMP/VASP 既能基于有限温度局部密度近似（自由能作为变分量），也能使用高效的矩阵对角化方案和有效 Pulay 混合对每个 MD 步骤的瞬时电子基态做精确评估。VASP 中，电子与离子间的相互作用使用超软赝势（Ultrasoft Vanderbilt Pseudopotentials，US-PP）或投影缀加波（Projector Augmented Wave，PAW）等方法描述。这两种方式都能有效减少过渡金属和第一行元素的每个原子所需的平面波数量。因此力和应力可以很容易地用 VAMP/VASP 计算，并用于将原子松弛到其瞬时基态。

英特尔® oneAPI Base 工具套件和 HPC 工具套件为 VASP 提供了良好的编译环境，并可使其在各类模拟场景中发挥更大效能。下文将提供一个在 Linux 平台上，基于英特尔® oneAPI Base 工具套件和 HPC 工具套件编译并运行 VASP 的示例，文中所描述的 VASP 版本为 6.2.0，使用者也可参考本文对更高版本的 VASP 进行编译使用。本文默认使用英特尔® MPI 库以及英特尔® 编译器。

更多 VASP 的信息，请访问 VASP 主页：https://www.vaspweb.org/
本文涉及的英特尔® oneAPI 基础包和 HPC 工具套件，请访问英特尔官网下载：https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html

■ 编译 VASP 准备工作

对于多节点的场景，最优的编译和运行选项很大程度取决于集群的配置。使用者可使用 Charm++ 的 MPI 后端（也可以使用 OFI），并使用英特尔® MPI 启动命令。

 # 步骤 1

使用以下命令解压缩 VASP 文件，其将创建一个 vasp.6.2.0 目录：

```
$tar –xvzf vasp.6.2.0.tgz
```

 # 步骤 2

通过运行以下命令来设置英特尔软件工具的环境变量，假设使用 64 位英特尔® 架构平台的默认安装和构建路径：

```
$source /opt/intel/oneapi/setvars.sh
```

■ 编译 VASP

 # 步骤 1

编译 libfftw3xf_intel.a，这是一个经高度优化的 FFTw（Fastest Fourier Transform in the west，一种快速傅里叶变换库）性能库，可有效加快 VASP 所涉及的 FFTw 工作负载。首先将目录更改为英特尔® oneMKL fftw3xf 库，并在其中对 fftw3xf 进行编译。编译完成后，将在同目录下得到 libfftw3xf_intel.a。

```
cd /opt/intel/oneapi/mkl/latest/interfaces/fftw3xf
make libintel64
```

 # 步骤 2

切换目录至 vasp.6.2.0，并复制 arch/makefile.include.linux_intel 文件至当前目录。

```
cd vasp.6.2.0
cp arch/makefile.include.linux_intel ./makefile.include
```

 # 步骤 3

编辑 makefile.include 文件，链接英特尔® oneMKL 中的 FFTw 库。

```
OBJECTS = fftmpiw.o fftmpi_map.o fft3dlib.o fftw3d.o  /opt/intel/oneapi/mkl/latest/interfaces/fftw3xf/libfftw3xf_intel.a
```

 # 步骤 4

检查 Fortran 和 C++ 编译器中，命令是否正确分配给：mpiifort、icc 和 icpc

```
FC      = mpiifort
FCL     = mpiifort -mkl=sequential
.....
.....
CC_LIB  = icc
.....
.....
CXX_PARS  = icpc
```

 # 步骤 5

检查 Fortran 标记部分。

```
FFLAGS   = -assume byterecl -w -xHOST
```

 # 步骤 6

可使用 -xHOST 启用 SIMD 指令。检查英特尔® oneMKL 中的 MKL 部分，包括用于 VASP 的 oneMKL BLAS、LAPACK、FFT、BLACS、ScaLAPACK 的函数链接。

```
MKL_PATH  = $(MKLROOT)/lib/intel64
BLAS      =
LAPACK    =
BLACS     = -lmkl_blacs_intelmpi_lp64
SCALAPACK = $(MKL_PATH)/libmkl_scalapack_lp64.a $(BLACS)
```

 # 步骤 7

使用以下命令编译 VASP，其将在 bin 目录中创建 vasp_std, vasp_gam, vasp_ncl 这几项可执行文件。

```
$make std gam ncl
```

■ 运行 VASP

 # 步骤 1

可通过配置参数的 mpiexec 命令来运行 VASP。例如可按以下代码所示配置工作负载（运行 48 个进程），并在 machinefile 中指明主机名。

```
$mpiexec.hydra –np 48 -f machinefile ./vasp_std
```

 # 步骤 2

检查 VASP 是否与英特尔® oneMKL 成功链接，可在 bin/vsp 中运行 ldd，如以下代码所示。

```
[vasp.6.2.0]$ ldd bin/vasp_std
```

检查显示结果：

```
linux-vdso.so.1 =>  (0x00007ffff3bd2000)
libmkl_intel_lp64.so.1 => /opt/intel/oneAPI/2021.2/mkl/2021.2.0/lib/intel64/libmkl_intel_lp64.so.1 (0x00002ac6f2a46000)
libmkl_cdft_core.so.1 => /opt/intel/oneAPI/2021.2/mkl/2021.2.0/lib/intel64/libmkl_cdft_core.so.1 (0x00002ac6f37ab000)
libmkl_scalapack_lp64.so.1 => /opt/intel/oneAPI/2021.2/mkl/2021.2.0/lib/intel64/libmkl_scalapack_lp64.so.1
(0x00002ac6f39d3000)
libmkl_blacs_intelmpi_lp64.so.1 => /opt/intel/oneAPI/2021.2/mkl/2021.2.0/lib/intel64/libmkl_blacs_intelmpi_lp64.so.1
(0x00002ac6f42fe000)
libmkl_sequential.so.1 => /opt/intel/oneAPI/2021.2/mkl/2021.2.0/lib/intel64/libmkl_sequential.so.1 (0x00002ac6f4544000)
libmkl_core.so.1 => /opt/intel/oneAPI/2021.2/mkl/2021.2.0/lib/intel64/libmkl_core.so.1 (0x00002ac6f6142000)
libiomp5.so => /opt/intel/oneAPI/2021.2/compiler/2021.2.0/linux/compiler/lib/intel64_lin/libiomp5.so (0x00002ac6ff6dd000)
.....
libmpifort.so.12 => /opt/intel/oneAPI/2021.2/mpi/2021.2.0//lib/libmpifort.so.12 (0x00002ac6ffdfb000)
libmpi.so.12 => /opt/intel/oneAPI/2021.2/mpi/2021.2.0//lib/release/libmpi.so.12 (0x00002ac7001b9000)
.....
.....
libfabric.so.1 => /opt/intel/oneAPI/2021.2/mpi/2021.2.0//libfabric/lib/libfabric.so.1 (0x00002ac70238d000)
```

面向 CP2K 的优化

CP2K 是一款功能齐备、性能强大的分子动力学模拟软件，其基于密度泛函理论（Density functional theory，DFT），为不同的建模方法（例如使用混合高斯的 DFT、使用平面波方法的 GPW 和 GAPW 等）提供了通用框架，可用于计算固态、液体、分子和生物体系的模拟与研究。作为一个开源项目，使用者可从官方网站（http://www.cp2k.org）下载相应版本的源代码。

通过选择适当的、可选的库和工具加入到 CP2K 的编译中，可以有效提升 CP2K 的工作效能。这些库包括：

- 英特尔®oneMKL, 将用到其中的 LAPACK/BLAS和ScaLAPACK库;
- FFTw（Fastest Fourier Transform in the west，一种快速傅里叶变换库）库;
- LIBXSMM（一个用于专门的密集和稀疏矩阵运算以及深度学习的库）;
- LIBINT(一个用于整数操作的库, 所使用版本可根据 CP2K 调整)
- PLUMED（一个包含增的采样工具、可用于分子动力学模拟数据分析等功能的库，建议使用 2.x 版本）
- LIBXC(一个面向密度泛函理论的交换相关泛函库, 建议使用 4.x 版本）
- ELPA（一个为对称矩阵提供高效、高可扩展的直接本征解算器的库，所使用版本可根据 CP2K 调整）

下文中将给出一个引入英特尔® MPI、英特尔® oneMKL、LIBXSMM、ELPA、LIBXC 和 LIBINT 等库的 MPI / OpenMP 混合版本的编译方法供使用者参考，该编译方法可使用英特尔® 编译器完成。

■ 基于各个优化库编译 CP2K

 # 步骤 1

安装英特尔® MPI、英特尔® oneMKL。

```
Source /opt/intel/oneapi/setvars.sh
```

 # 步骤 2

下载、编译并安装 ELPA（使用英特尔® 编译器）。

```
cd $HOME
echo "wget --content-disposition --no-check-certificate https://elpa.mpcdf.mpg.de/software/tarball-archive/Releases/2020.11.001/elpa-2020.11.001.tar.gz"
wget --content-disposition --no-check-certificate https://www.cp2k.org/static/downloads/elpa-2020.11.001.tar.gz
tar xvf elpa-2020.11.001.tar.gz

cd elpa-2020.11.001
wget --content-disposition --no-check-certificate https://github.com/hfp/xconfigure/raw/master/configure-get.sh
chmod +x configure-get.sh
./configure-get.sh elpa

./configure-elpa-skx-omp.sh

make -j
make install
make clean
```

 # 步骤 3

下载、编译并安装 LIBINT（使用英特尔® 编译器）。

```
cd $HOME
curl -s https://api.github.com/repos/cp2k/libint-cp2k/releases/latest \
| grep "browser_download_url" | grep "lmax-6" \
| sed 's/.*: \"\(.*[^\"]\)\".*/url \1/' \
| curl -LOK-
tar xvf libint-v2.6.0-cp2k-lmax-6.tgz

cd libint-v2.6.0-cp2k-lmax-6
wget --content-disposition --no-check-certificate https://github.com/hfp/xconfigure/raw/master/configure-get.sh
chmod +x configure-get.sh
./configure-get.sh libint

./configure-libint-skx.sh

make -j
make install
make distclean
```

 # 步骤 4

下载、编译并安装 LIBXC（使用英特尔® 编译器）。

```
cd $HOME
wget --content-disposition --no-check-certificate https://gitlab.com/libxc/libxc/-/archive/4.3.4/libxc-4.3.4.tar.bz2
tar xvf libxc-4.3.4.tar.bz2

cd libxc-4.3.4
wget --content-disposition --no-check-certificate https://github.com/hfp/xconfigure/raw/master/configure-get.sh
chmod +x configure-get.sh
./configure-get.sh libxc

./configure-libxc-skx.sh

make -j
make install
make distclean
```

 # 步骤 5

下载、编译并安装 PLUMED（使用英特尔® 编译器）。

```
cd $HOME
wget --content-disposition --no-check-certificate https://github.com/plumed/plumed2/archive/v2.6.1.tar.gz
tar xvf v2.6.1.tar.gz

cd plumed2-2.6.1
wget --content-disposition --no-check-certificate https://github.com/hfp/xconfigure/raw/master/configure-get.sh
chmod +x configure-get.sh
./configure-get.sh plumed

./configure-plumed-skx.sh

make -j
make install
make distclean
```

 # 步骤 6

下载 LIBXSMM，LIBXSMM 的编译在步骤 7 完成。

```
cd $HOME
wget --content-disposition --no-check-certificate https://github.com/hfp/libxsmm/archive/1.16.1.tar.gz
tar xvf 1.16.1.tar.gz
```

 # 步骤 7

下载并编译 CP2K 的 PSMP 变体，这里需要从 GitHub 重新下载 ARCH 文件。如未找到英特尔® oneMKL，则可将密钥 MKLROOT=/path/to/MKL 添加到 Make 的命令行中（详见步骤 1 说明）。要选择不同的 MPI 实现，可尝试使用命令如 MKL_MPIRTL=openmpi。编译完成后，可得到 CP2K 的可执行文件，可使用 1 exe/Linux-x86-64-intelx/cp2k.psmp 命令检查。

```
cd $HOME
wget https://github.com/cp2k/cp2k/releases/download/v7.1.0/cp2k-7.1.tar.bz2
tar xvf cp2k-7.1.tar.bz2

cd cp2k-7.1
wget --content-disposition --no-check-certificate https://github.com/hfp/xconfigure/raw/master/configure-get.sh
chmod +x configure-get.sh
./configure-get.sh cp2k

rm -rf exe lib obj

make ARCH=Linux-x86-64-intelx VERSION=psmp AVX=3 \
  LIBINTROOT=$HOME/libint/intel-skx \
  LIBXCROOT=$HOME/libxc/intel-skx \
  ELPAROOT=$HOME/elpa/intel-skx-omp -j
```

在编译过程中使用英特尔 ARCH 文件，可以帮助使用者避免重复编写 ARCH 文件，英特尔 ARCH 文件能自动为编译中所需的英特尔库获取正确的路径。当英特尔工具是源代码时，这些路径是通过使用环境变量设置来确定的。同时，当上述各个库的 LIBXSMMMROOT、LIBINTROOT、LIBXCROOT 和 ELPAROOT 位于主目录中，或与 CP2K 目录在一起，英特尔 ARCH 文件也能及时发现它们。使用者可以手动下载上述英特尔 ARCH 文件：

```
cd cp2k-6.1.0/arch
wget https://github.com/hfp/cp2k/raw/master/arch/Linux-x86-64-intelx.arch
wget https://github.com/hfp/cp2k/raw/master/arch/Linux-x86-64-intelx.popt
wget https://github.com/hfp/cp2k/raw/master/arch/Linux-x86-64-intelx.psmp
wget https://github.com/hfp/cp2k/raw/master/arch/Linux-x86-64-intelx.sopt
wget https://github.com/hfp/cp2k/raw/master/arch/Linux-x86-64-intelx.ssmp
```

■ 运行 CP2K

当 CP2K 运行在多节点的场景中时，如何配置进程与线程就会对性能产生较大影响。使用者可以根据实际情况来对方案进行编排，例如当工作负载对内存占用不大时，对较低节点使用高秩计数（High Rank Count）可能会产生更佳结果。而当工作负载非常占用内存时，尤其是工作负载需要复制大量内存而不是按照秩（Rank）的数量进行分区时，需要适当降低每个节点的 MPI 秩的数量，并对较低节点使用低秩计数（Low Rank Count）。

值得一提的是，在大多数情况下 CP2K 更喜欢总秩计数（Total Rank Count）为平方数，这在对秩 / 线程组合调优时有着更大的复杂性。上述调优可参阅 MPI/OpenMP 混合的脚本文档 plan.sh，这一文档设定并解释了相关的 MPI 环境变量。

下文展示了在一个英特尔® 超线程技术开启且每路处理器有 24 个物理内核，即共有 96 个线程的双路系统中应该如何规划运行 CP2K 的 PSMP 二进制文件。在单节点上，对于 16 个秩和每个秩 6 个线程的情况，可设置为：1x16x6。CP2K 运行命令如下：

```
mpirun -np 16 \
  -genv I_MPI_PIN_DOMAIN=auto -genv I_MPI_PIN_ORDER=bunch \
  -genv OMP_PLACES=threads -genv OMP_PROC_BIND=SPREAD \
  -genv OMP_NUM_THREADS=6 \
  exe/Linux-x86-64-intelx/cp2k.psmp workload.inp
```

对于 8 个节点的 MPI 命令，可在 plan.sh 中为每个节点设置 8 个秩，每个秩有 12 个线程的情况：8x8x12。CP2K 运行命令如下：

```
mpirun -perhost 8 -host node1,node2,node3,node4,node5,node6,node7,node8 \
  -genv I_MPI_PIN_DOMAIN=auto -genv I_MPI_PIN_ORDER=bunch \
  -genv OMP_PLACES=threads -genv OMP_PROC_BIND=SPREAD \
  -genv OMP_NUM_THREADS=12 -genv I_MPI_DEBUG=4 \
  exe/Linux-x86-64-intelx/cp2k.psmp workload.inp
```

■ CP2K 性能调优

 # 步骤 1

在 CP2K 中，可以通过调整 MPI 通信模式来对其实现性能调优，对于使用英特尔® MPI 时，使用者可尝试下列配置：

```
export I_MPI_COLL_INTRANODE=pt2pt
export I_MPI_ADJUST_REDUCE=1
export I_MPI_ADJUST_BCAST=1
```

 # 步骤 2

同时在 CP2K 大规模运行时，也可在启动时加入以下命令来提升性能：

```
export I_MPI_DYNAMIC_CONNECTION=1
```

 # 步骤 3

通常来说，英特尔® MPI 会为使用者设定好 InfiniBand。如果使用者需要进行个性化设置，例如可使用 mpirun-RDMA 来显式使用基于 RDMA 的 InfiniBand，可通过以下环境变量的设置来实现：

```
echo "'mpirun -rdma' and/or environment variables for InfiniBand"
export I_MPI_FABRICS=shm:ofi
```

 # 步骤 4

方便地进行日志输出也是 CP2K 运行时必要的调优方向。使用者可以使用一个信息脚本（info.sh）来尝试呈现一个表（所有结果的摘要），该表是从日志文件生成的（使用 tee，或者依赖于作业调度程序的输出），并只支持某些文件扩展名（.txt、.log）。如以下示例所示：

```
./run-cp2k.sh | tee cp2k-h2o64-2x32x2.txt
ls -1 *.txt
cp2k-h2o64-2x32x2.txt
cp2k-h2o64-4x16x2.txt

./info.sh [-best] /path/to/logs-or-cwd
H2O-64        Nodes R/N T/R Cases/d Seconds
cp2k-h2o64-2x32x2 2 2    32  4   807 107.237
cp2k-h2o64-4x16x2 4    16  8   872 99.962
```

■ CP2K 的部署

堆内存的动态分配通常需要全局核算，最终在应用程序的共享内存并行区域中产生开销。对于这种情况，可采用专门的分配策略。为使用这样的策略，可以在编译时或应用程序运行时使用内存分配包装器来替换默认的内存分配。要使用英特尔® TBB 技术的 malloc 代理，需要在 CP2K 的编译时设置 TBBMALLOC=1 键值对（默认值：TBBMALLOC=0）。

生命科学与高性能的科学计算平台

在生命科学研究领域，信息化技术的运用正不断走向深入。借助计算机工具，科学家与研究机构可以高效地在基因组学（Genomics）分析、蛋白质组学（Proteomics）研究等方向上获得必要的生物信息储存、检索和分析助力，从而加速药物设计、疫苗研发、疾病筛查以及精准医疗服务等方面的研发和落地进程。

随着生命科学研究的深入，其面对的数据量和所需的算力也逐渐达到了一个惊人的规模。例如在高通量测序（High-throughput sequencing，HTS）技术中，单次测序生成的有效数据可达数 GB 之多。同时，复杂生物系统模型构建、病毒基因同源性分析、蛋白质动力学特性探索等深层次研究，更多地引入了 AI 能力，让处理系统的算力、内存以及存储能力都面临严峻的挑战。

因此，科学计算平台正在生命科学领域中发挥越来越大的作用。一般而言，生命科学相关应用软件对计算平台的需要主要包括：

- **强劲充沛的算力支持**：生命科学计算任务往往需要大量的高精度浮点计算能力，例如在冷冻电镜图像的处理和重构上，需要执行海量的快速傅立叶（FFT）等计算；
- **更高的内存带宽性能**：在生命科学计算任务中，所需计算和存储的数据样本以及模型数据非常庞大（甚至达到 TB 级别）且数据读写频繁，需要为之提供匹配的内存带宽性能；
- **集群化并行处理能力**：并行计算也是提升生命科学计算任务效率的重要方法，其可以有效缩短任务运行时间。

英特尔一直以来都在通过其不断迭代更新的处理器平台和丰富全面的软件栈和加速库，为生命科学领域的软件提供强劲的算力支持和专门的性能优化。下文中将就 Relion、基因组分析软件栈等常见生命科学软件，在基于英特尔® 架构的平台上的优化编译和运行展开介绍。

面向基于英特尔® 架构平台的生命科学软件优化

面向英特尔® 架构平台的 Relion 优化编译与运行

冷冻电子显微镜（Cryo-electron microscopy, Cryo-EM）是一种在低温下使用透射电子显微镜观察冷冻固定样本的显微技术，该技术是结构生物学研究的重要手段之一。冷冻电镜的工作流程中，通常需要对海量图像进行分析和细化，巨大的计算量需要先进的科学计算平台与高效的软件、算法相配合才能完成。

作为一个开源应用套件，由 MRC 分子生物学实验室的 Sjors Scheres 小组开发的 Relion，是专门为冷冻电镜而设计的图像处理软件。该软件加入了多种算法来完成 2D 分类、3D 分类以及 3D 细化等功能，可帮助生物领域的研究者，通过对冷冻电镜数据的单颗粒分析来优化大分子结构。

Relion 可在多种硬件架构上以不同的方式处理各种数据集。英特尔也以其在处理器平台和软件加速性能上的独特优势，为 Relion 的工作流程提供了强有力的支撑。从第三代、第四代英特尔® 至强® 可扩展处理器到英特尔® 至强® CPU Max 系列，英特尔的一系列处理器平台提供了更为出色的、具有内置 AI 加速功能的工作负载优化平台。其中，能够对 Relion 性能提升提供助力的包括但不限于：

- 增强的基础算力性能；
- 增强的内存速度和容量（DDR4/DDR5 内存，HBM 内存）；
- 增强型英特尔® SST

下文将简述如何在基于英特尔® 架构的平台上编译并运行 Relion，并获得更优性能的过程。借助基于英特尔® 架构的平台，使用者可以对处理器、内存、存储设备和网络做整体优化配置 / 设置，以获得更佳的工作效能。本文将以第三代英特尔® 至强® 可扩展处理器为例，但相关配置 / 设置经适当调整后，也可应用于其它基于英特尔® 架构的理器平台。

■ 系统配置 / 设置调优

无论是操作系统，还是 Relion 软件的配置 / 设置，默认都是基于通用应用程序的模式设计，起初都不会基于性能优化的目的进行特别设置，因此有针对性地开展软件调优配置 / 设置是必要的。

本质上，Relion 是由一系列用于分阶段冷冻电镜（Cryo-EM）数据处理的应用程序组成。整个工作流程如图 2-4-1 所示，在每个步骤之间，Relion 会从磁盘读取上一个步骤的结果，并在当前步骤结束时将结果写入磁盘，通常需要保留中间结果数据（以允许从中间过程中恢复数据），各步骤也可循环迭代地执行。所有这些步骤和中间结果数据都会生成大量大文件，这些大文件的处理时间和存储要求也各有不同。一旦所有步骤完成，系统通常会归档出一系列 100 GB 左右的数据包用以重新分析。

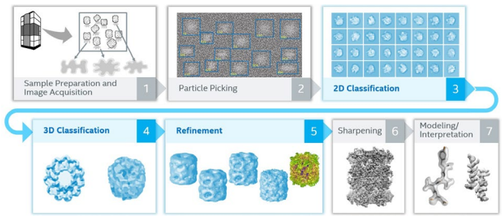


图 2-4-1 Relion 工艺流程阶段的简化视图

上述处理步骤都涉及计算密集型和磁盘密集型操作。因此，建议面向每个电镜的最小处理集群包括：

- 每个节点有 2 个双路处理器，每个处理器具有 48 至 64 个内核，并配备 256 至 384GB（或更多）内存；
- 一个 2PB 磁盘存储空间的并行文件系统；
- 节点和并行文件系统之间，由高速集群网络连接；
- 上述配置应随着电镜的增加而线性增加。

同时，也建议处理集群的服务器做以下配置 / 设置。

内置增强技术配置 / 设置：

- 英特尔® 睿频加速技术：这一技术允许处理器在低于当前功率或温度规格的情况下自动提高频率。
- 增强型英特尔® SST：这一技术允许系统动态调整处理器电压和内核频率，从而降低平均功耗和平均发热量。为了使英特尔® 睿频加速技术可用，必须启用增强型英特尔® SST。
- 英特尔® 超线程技术：这一技术允许多线程应用在每个处理器内核内并行执行两个线程，即线程可运行在两倍于物理核的逻辑核上。

内存配置 / 设置：

- 建议每个节点配备 256GB 以上内存。每个内存通道至少需要在主板上安装 1 个内存条（Dual Inline Memory Module，DIMM）。如果因数据争用而导致内存通道中没有 DIMM，可能会导致处理器利用率降低。

存储 / 磁盘配置 / 设置：

- Relion 处理的数据集往往极为庞大，通常需要通过多节点处理模式来提升性能，因此建议在集群文件系统（Cluster File System，CFS）上使用 Relion，CFS 系统已针对大小不同文件（如 Lustre）的读 / 写访问进行了优化。考虑到 Cryo-EM（冷冻电镜）工作负载的数据需求，建议 CFS 系统为每个显微镜至少配备 2PB 存储空间。

网络配置 / 设置：

- Relion 要发挥更佳性能，需要借助 MPI 在多个节点上进行数据处理。虽然 Relion 本身不会生成很多 MPI 消息，但 Relion 数据集和临时文件会消耗大量的磁盘空间，且这些文件在处理过程中会被所有节点访问。建议 Relion 部署在 100Gbps 或更快的网络环境中，使用企业级高速集群结构和并行文件系统。昂贵的冷冻电镜系统每天产生的数据以数 TB（Terabytes）计，且经常会被超额预订，需要更为高速可靠的数据传输、处理能力来挖掘这些稀缺资源的价值。

进一步的，为帮助 Relion 的使用者更方便地实现性能优化，英特尔与众多合作者一起，面向英特尔® 架构平台的特性，对图 2-4-1 中步骤 3 至步骤 5 进行了专门的优化。下文将就这些优化方法做简要描述。

■ 面向基于英特尔® 架构平台的编译及运行

Relion 中的 2D 分类、3D 分类以及 3D 细化算法已面向英特尔® AVX - 512 进行了优化（从 3.0 版本开始支持），在编译 Relion 打开相应标志位（ALTCPU=ON），并在运行时加入 --CPU 选项即可支持。这一优化能在所支持的处理器平台上获得显著性能提升。同时，与默认的 Linux 编译器相比，使用英特尔® 编译器也能带来额外的性能提升。

下文给出了一个编译过程示例，编译过程基于已集成入英特尔® oneAPI 工具套件完成，并使用了英特尔® MPI 库、英特尔® oneMKL 和英特尔® 编译器。

 # 步骤 1

下载 Relion（3.1.1 或更高版本）并设置编译器和库环境。

```
1. source /opt/intel/oneapi/setvars.sh
```

 # 步骤 2

从 Git 拉取源代码后，在 Relion 目录下编译一个基于英特尔® 至强® 可扩展处理器（支持英特尔® AVX - 512）的 Relion 版本。

```
1. $ mkdir build
2. $ cd build
3. $ cmake -DCMAKE_C_COMPILER=icc -DCMAKE_CXX_COMPILER=icpc -
   DMPI_C_COMPILER=mpiicc -DMPI_CXX_COMPILER=mpiicpc -D CMAKE_C_FLAGS="-O3 -ip -g -
   debug inline-debug-info -xCOMMON-AVX512 -qopt-report=5 -restrict " -D CMAKE_CXX_FLAGS="-O3 -
   ip -g -debug inline-debug-info -xCOMMON-AVX512 -qopt-report=5 -restrict " -DALTCPU=ON -
   DMKLFFT=ON -DGUI=OFF -D CMAKE_BUILD_TYPE=Release ..
4. $ make -j 20
5. $ cp bin/relion_refine_mpi ../relion_refine_mpi_3.1.1_AVX512
```

 # 步骤 3

启用英特尔® 超线程技术能使 Relion 在运行时获得更优的性能表现。同时，受限于 Relion 的数据流管理能力，以及工作任务执行并行性与延迟的限制，建议作业时的总服务器数量不超过 16 台。Relion 运行时，参数可按照以下方式设置：

在双路英特尔® 架构处理器平台上，每个节点配置 8 个 MPI 计算列。此时可设置每个计算列对应的线程数（--j）为逻辑内核总数（启用英特尔® 超线程技术）除以计算列数（8）；

- 可能的话，建议每个计算列对应的线程数（--j）大于 9；
- 当节点的集群内存小于 128GB 时，因适当减少计算列，避免出现内存瓶颈或数据交换阻塞；
- 将池（--pool）的数量设置为线程数（--j）的两倍左右，且不低于 30。

可以参考以下设置示例：

```
$ export OMP_SCHEDULE="dynamic"
$ export KMP_BLOCKTIME=0
$ export I_MPI_PIN_DOMAIN=<the total logical cores in the machine divided by the number of compute ranks per machine>
$ export I_MPI_DEBUG=5
$ export I_MPI_FABRICS=
```

 # 步骤 4

单节点示例：以在启用英特尔® 超线程技术的双路英特尔® 至强® 铂金 8358 处理器的单节点系统上运行 Relion 为例，处理器平台一共可提供 128 个逻辑内核。在每个节点部署 8 个计算列的情况下，每个列配置 16 个线程，以 40 的池大小运行。加上一个额外的用于列控制的附加列，命令序列如下所示：

```
$ export OMP_SCHEDULE="dynamic"
$ export KMP_BLOCKTIME=0
$ export I_MPI_PIN_DOMAIN=16
$ export I_MPI_DEBUG=5
$ export I_MPI_FABRICS=
$ mpirun -n 9 relion_refine_mpi <other flags and options> --j 16 --pool 40
```

 # 步骤 5

多节点示例：以在启用英特尔® 超线程技术的双路英特尔® 至强® 铂金 8380 处理器的 4 节点系统上运行 Relion 为例，处理器平台一共可提供 160 个逻辑内核。在每个节点部署 8 个计算列的情况下，每个列配置 20 个线程，以 40 的池大小运行。加上一个额外的用于列控制的附加列，命令序列如下所示：

```
$ export OMP_SCHEDULE="dynamic"
$ export KMP_BLOCKTIME=0
$ export I_MPI_PIN_DOMAIN=20
$ export I_MPI_DEBUG=5
$ export I_MPI_FABRICS=<appropriate fabrics for your installation>
$ mpirun -n 33 relion_refine_mpi <other flags and options> --j 20 --pool 40
```

■ 对优化编译的验证与最佳实践

为验证 Relion 在英特尔® 架构处理器平台上的优化编译效果，可以使用疟原虫核糖体数据集对其开展用于验证的基准测试。该数据集可以在 Github 的 Relion 基准测试页面中下载并运行（https://github.com/3dem/relion）。

 # 步骤 1

下载并解包疟原虫核糖体数据集后，转到所生成目录树的顶层会出现一个名为 emd_2660.map 的文件和一个名称为 Particles 的目录。在运行基准测试前，可能需要将要处理的数据文件从旧格式转换为新格式并确定基准时间。测试中，可以使用 /usr/bin/time 对二进制文件进行计时，或如以下示例中所用的，将 mpirun 命令内置于在脚本中。

```
#格式转换
$ <relion_source_tree/build/bin/>relion_convert_star --i Particles/shiny_2sets.star --o Particles/shiny_2sets_3.1.star
#确定基准时间
btime=`date +%s`
<run_RELION_as_per_below_via_mpirun>
etime=`date +%s`
diff=`expr $etime - $btime`
echo "RELION ... < information about the run parameters> completed in the following time (seconds): $diff "
```

 # 步骤 2

在启用英特尔® 超线程技术的双路英特尔® 至强® 铂金 8358 处理器的单节点系统上对疟原虫核糖体数据集进行计算。

```
$ mkdir Results
$ export OMP_SCHEDULE="dynamic"
$ export KMP_BLOCKTIME=0
$ export I_MPI_PIN_DOMAIN=16
$ export I_MPI_FABRICS=
$ btime=`date +%s`
$ mpirun -hostfile <one_node_host_file> -env OMP_SCHEDULE=dynamic -env KMP_BLOCKTIME=0 -perhost 9 -np 9 <binary_location>/relion_refine_mpi_3.1.1_AVX512 --i Particles/shiny_2sets_3.1.star --ref emd_2660.map:mrc --firstiter_cc --ini_high 60 --dont_combine_weights_via_disc --ctf --ctf_corrected_ref --tau2_fudge 4 --particle_diameter 360 --K 6 --flatten_solvent --zero_mask --oversampling 1 --healpix_order 2 --offset_range 5 --offset_step 2 --sym C1 --norm --scale --random_seed 0 --pool 40 --j 16 --iter 25 --cpu --o Results/cpu3d
$ etime=`date +%s`
$ diff=`expr $etime - $btime`
echo "RELION 3.1.1 completed 25 iterations of Plasmodium Ribosome using 9 ranks and 16 threads per rank and a pool size of 40 on a single Intel® Xeon® 8358 Scalable Processor node in the following time (seconds): $diff "
```

 # 步骤 3

在启用英特尔® 超线程技术的双路英特尔® 至强® 铂金 8358 处理器的 4 节点系统上对疟原虫核糖体数据集进行计算。

```
$ mkdir Results
$ export OMP_SCHEDULE="dynamic"
$ export KMP_BLOCKTIME=0
$ export I_MPI_PIN_DOMAIN=16
$ export I_MPI_DEBUG=5
$ export I_MPI_FABRICS=
$ btime=`date +%s`
$ mpirun -hostfile <four_node_host_file> -env OMP_SCHEDULE=dynamic -env KMP_BLOCKTIME=0 -perhost 8 -np 33 <binay_location>/relion_refine_mpi_3.1.1_AVX512 --i Particles/shiny_2sets_3.1.star --ref emd_2660.map:mrc --firstiter_cc --ini_high 60 --dont_combine_weights_via_disc --ctf --ctf_corrected_ref --tau2_fudge 4 --particle_diameter 360 --K 6 --flatten_solvent --zero_mask --oversampling 1 --healpix_order 2 --offset_range 5 --offset_step 2 --sym C1 --norm --scale --random_seed 0 --pool 40 --j 16 --iter 25 --cpu --o Results/cpu3d
$ etime=`date +%s`
$ diff=`expr $etime - $btime`
echo "RELION 3.1.1 completed 25 iterations of Plasmodium Ribosome using 33 ranks and 16 threads per rank and a pool size of 40 on four Intel® Xeon® 8358 Scalable Processor nodes in the following time (seconds): $diff "
```


 # 步骤 4

与参考值做比较：如果需要将经优化编译的（使用 --cpu 选项）Relion 运行结果与使用 Chimera 等工具的原始 Relion 算法进行比较，只需要将结果以不同目录或不同名称输出即可。使用者可以将两种结果的 *.mrc 文件可以加载到 Chimera 中并覆盖就能了解结果的差异，使用者也可以运行 3D 细化来比较相关曲线的差异。

面向基因组分析的英特尔® 精选解决方案

作为生物学研究领域最重要的方向之一，基因组学（Genomics）是对生物体所有基因做集体表征、定量研究，并面向不同基因组开展比较研究的学科。基于基因组分析，研究者能够进一步探索生物体的基因组结构与功能以及基因间的关系，从而帮助人们更好地了解生物多样性、改良生物品种、资源保护，以及为疾病治疗提供理论指导。因此，面向不同基因组学分支的基因组分析任务，正在医疗、生物等相关领域获得越来越多的关注。

■ 基因组分析软件栈与面向基因组分析的英特尔® 精选解决方案

基因组分析是一项复杂且需要消耗巨量算力的工作。目前广泛用于基因组分析的软件栈包括（以下软件建议选择较新的版本）：

- **基因组分析工具套件**（Genomic Analysis Toolkit，GATK）：用于从数据中分析变异信息，是目前最主流的 snp calling 软件之一；
- Cromwell：用于科学工作流程的管理系统，用于跟踪工作流程并将结果存储于 MariaDB 数据库；
- BWA：用于对照大的参考基因组（如人类基因组）绘制低分化序列；
- Picard：用于操作高通量测序数据（基于不同格式）的工具；
- VerifyBAMID2：用于验证样本基因组数据是否与已知基因型匹配，并可检测样本污染；
- Samtools：用于与高通量测序数据交互（读取、写入、编辑、索引或查看），读写 BCF2、VCF 等文件，以及调用、过滤或汇总 SNP 和短 indel 序列变体；
- 20K Throughput Run：是一个快速基准测试程序，用于确保计算集群中的重要功能是否配置正确；
- **英特尔® 系统配置实用程序**（Intel® System Configuration Utility）**（可选）**：可用于保存和恢复系统 BIOS 和管理固件设置的命令行实用程序。

在使用上述基因组分析软件组合时，以下软件或开发工具套件也是不可或缺的：

- 用于版本跟踪、修订的版本控制系统 Git；
- Java、JRE（Java Runtime Environment，Java 运行环境）及 Java SDK；
- Python（建议 3.6.2 以上版本）及 Conda 包管理器；
- Slurm Workload Manager，提供了一种调度作业并为这些作业分配集群资源的方法。有关 Slurm 的详细说明，请访问：https://slurm.schedmd.com/documentation.html.
- 编译 Cromwell 所必需的 sbt
- 用于 Cromwell 持久存储的 MariaDB
- 用于 R、GATK 等软件工具的图形生成的 Rscript、gsalib、ggplot2 等。

通过面向基因组分析的英特尔® 精选解决方案，英特尔为上述软件工具和开发套件所需的算力提供了更强有力的支持。如图 2-4-2 所示，面向基因组分析的英特尔® 精选解决方案架构中，底层是由第三代、第四代英特尔® 至强® 可扩展处理器，英特尔® 至强® CPU Max 系列等为核心的硬件基础设施。英特尔也以其在处理器平台和软件加速性能上的独特优势，为基因组分析的工作流程提供了强有力的支撑。一系列处理器平台提供了业界领先的、具有内置 AI 加速功能的工作负载优化平台。其中，对基因组分析性能提升有着助力的包括但不限于：

- 增强的基础性能；
- 增强的内存速度和容量（DDR4/DDR5 内存，HBM 内存）；
- 借助英特尔® AVX - 512 实现优化的英特尔基因组学内核库。

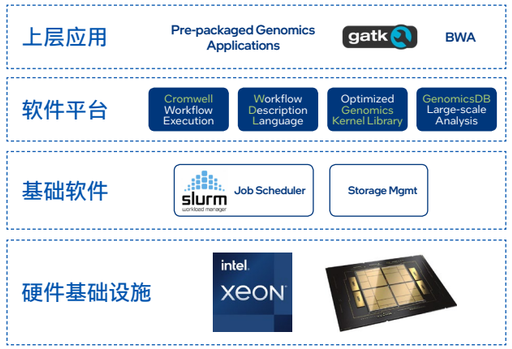


图 2-4-2 面向基因组分析的英特尔® 精选解决方案架构

这一架构中，使用者也可根据需求，开展进一步的优化。例如对于英特尔® 至强® CPU Max 系列，使用者可参考以下网址内容进行调优：https://www.intel.com/content/www/us/en/content-details/782255/intel-xeon-cpu-max-series-configuration-and-tuning-guide.html

在硬件基础设施层之上，是由一系列软件构成的基础软件层、软件平台层以及上层应用层。研究者们可以基于这一架构，可以使用 GATK 等工具及 WDL 脚本来处理 DNA、RNA 数据，使用 Cromwell 管理工作流程，使用 Slurm 调度作业并为这些作业分配集群资源等等。

■ 基因组分析软件调优

在基因组分析的工作中，软件配置调整是至关重要的一步。究其原因，是因为操作系统和基因组学分析软件都是面向通用的应用程序模式而设计的，需要面向基因组分析的实际需求开展优化以获得更佳性能。下文将基于面向基因组分析的英特尔® 精选解决方案，提供方案中主要软件的优化编译、配置和运行建议。

■ 安装配置 Slurm Workload Manager

 # 步骤 1

Slurm Workload Manager 为基因组分析工作流程提供了一种作业流程调配和集群资源分配的方法。可安装以下步骤开始安装 Slurm。

- 创建一个被授予不受限制的 SSH 访问权限的用户组：
- 将“英特尔® 群集检测器”专用用户添加到信任列表（即白名单）。此用户帐户应能在资源管理器作业内外运行群集检查器。
- 创建 Slurm 用户帐户：
- 安装 Slurm 服务端软件包
- 升级 Warewulf 文件

```
groupadd sshallowlist
usermod -aG sshallowlist clck
useradd --system --shell=/sbin/nologin slurm
dnf -y install ohpc-slurm-server
wwsh file sync
```

 # 步骤 2

更新节点资源信息：在 Slurm 配置文件中更新计算节点的节点名称、处理器的属性以及与计算集群相关联的 Slurm 分区或队列。

- NodeName：反映计算节点的名称及其各自功能的定义；
- Sockets：定义计算节点中的套接字数量；
- CoresPerSocket：定义每个套接字的处理器内核数；
- ThreadsPerCore：定义可在处理器内核上并行执行的线程数；
- PartitionName：定义了 Slurm 分区或队列，在这些分区或队列中分配了计算名称；
- Nodes：定义了配置给给定分区的计算节点；
- Default：定义默认分区。

同时，完成以下任务并为计算集群创建 Slurm 配置文件：

- 使用 openHPC 模板创建一个新的 Slurm 配置文件；
- 将前端主机的主机名作为 ControlMachine 添加到 Slurm 配置文件中；
- 确保 SLURM 控制后台程序在节点关闭后，通过有效配置进行注册后该节点可再次使用；
- 更新 NodeName 定义以映射硬件功能；
- 更新 PartitionName 定义。

Slurm 配置文件 slurm.conf 的创建、更新、导入和启用命令示例如下所示：

```
cp /etc/slurm/slurm.conf.ohpc /etc/slurm/slurm.conf
sed -i "s/^\(NodeName.*\)/#\1/" /etc/slurm/slurm.conf
echo "NodeName=${compute_prefix}[${first_node}-${last_node}] Sockets=${num_sockets} \
CoresPerSocket=${num_cores} ThreadsPerCore=2 State=UNKNOWN" >> /etc/slurm/slurm.conf
sed -i "s/^ControlMachine=.*/ControlMachine=${frontend_name}/" /etc/slurm/slurm.conf
sed -i "s/^\(PartitionName.*\)/#\1/" /etc/slurm/slurm.conf
sed -i "s/^\(ReturnToService.*\)/#\1\nReturnToService=2/" /etc/slurm/slurm.conf
sed -i "s/^\(SelectTypeParameters=.*\)/#\1/" /etc/slurm/slurm.conf
sed -i "s/^\(JobCompType=jobcomp\/none\)/#\1/" /etc/slurm/slurm.conf
cat >> /etc/slurm/slurm.conf << EOFslurm
PartitionName=xeon Nodes=${compute_prefix}[${first_node}-${last_node}] Default=YES MaxTime=24:00:00 State=UP
PartitionName=cluster Nodes=${compute_prefix}[${first_node}-${last_node}] Default=NO MaxTime=24:00:00 \
State=UP
EOFslurm
wwsh -y file import /etc/slurm/slurm.conf
sed -i "s/^files\(.*\)/files\1, slurm.conf, munge.key/" /etc/warewulf/defaults/provision.conf
syssystemctl enable slurmctld.service
```

 # 步骤 3

将 Slurm 客户端安装到计算节点映像中，命令示例如下所示：

```
echo "+ : root : ALL">>$CHROOT/etc/security/access.conf
echo "+ : sshallowlist : ALL">>$CHROOT/etc/security/access.conf echo "- : ALL : ALL">>$CHROOT/etc/security/access.conf
echo "account sufficient pam_access.so" >> $CHROOT/etc/pam.d/sshd
wwvnfs --chroot $CHROOT --hybridize
```

 # 步骤 4

完成 Slurm 配置，命令示例如下所示：

```
cat >> /etc/environment << EOFSLURM export SLURM_OVERLAP=1 EOFSLURM
systemctl restart munge
pdsh -w c[01-XX] reboot
scontrol reconfig
scontrol update NodeName=c[01-XX] State=Idle
[root@frontend ~]# sinfo
```

 # 步骤 5

一旦资源管理器投入生产，使用者就能以此运行作业。可以通过一个具有标准权限的测试账号来进行测试。例如，该账号将不被允许使用 SSH 与 Slurm 作业之外的计算节点进行交互。通过资源管理器以交互方式编译和执行 helloworld 应用程序。

```
useradd -m test
passwd test
wwsh file resync
su – test
module load oneapi
mpicc -o test /opt/intel/oneapi/mpi/latest/test/test.c
[test@frontend ~]# srun --mpi=pmi2 --partition=xeon -N 4 -n 4 /home/test/test
Hello world: rank 0 of 4 running on c01.cluster
```


■ 安装基因组分析软件栈

 # 步骤 1

检查基因组分析计算集群的安装环境。

```
pdsh -w frontend,c0[1-4] "su - cromwell -c 'ulimit -n'"
[root@frontend ~]# pdsh -w c0[1-4] "stat -c '%A %U:%G %m' /genomics_local"
c01: drwxr-xr-x cromwell:cromwell /genomics_local
c03: drwxr-xr-x cromwell:cromwell /genomics_local
c04: drwxr-xr-x cromwell:cromwell /genomics_local
c02: drwxr-xr-x cromwell:cromwell /genomics_local
```

 # 步骤 2

配置并运行 MariaDB，Cromwell 使用 MariaDB 来存储工作流程中的信息。

```
systemctl enable mariadb.service
systemctl enable httpd.service
systemctl restart mariadb
systemctl restart httpd
#Edit this file: /etc/warewulf/database-root.conf
database password = <new_password>
#Save and exit the file
/usr/bin/mysql_secure_installation
wwinit DATASTORE
mysql -h localhost -u root -p
#Create a new database named "cromwell"
MariaDB [(none)]> CREATE DATABASE cromwell;
MariaDB [(none)]> CREATE USER 'cromwell'@'localhost' IDENTIFIED BY 'cromwell';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON `cromwell`.* TO 'cromwell'@'localhost';
MariaDB [(none)]> FLUSH PRIVILEGES;
MariaDB [(none)]> exit
```

 # 步骤 3

安装 Cromwell 工作流管理系统，并将其配置为使用计算节点上的本地暂存设备。

```
sed -i "s/^\(import\ java.util.UUID\)/\/\/\1/" \
backend/src/main/scala/cromwell/backend/RuntimeEnvironment.scala
sed -i '23,27d' \ backend/src/main/scala/cromwell/backend/RuntimeEnvironment.scala
sed -i '23i \\\\\ def tempPath: String = \"/genomics_local\" " \ backend/src/main/scala/cromwell/backend/RuntimeEnvironment.scala
sed 's/\(\s*\)export _JAVA_OPTIONS."\) "/\1\/tmp\.\$\$\$\$\n\1 "/" \ backend/src/main/scala/cromwell/backend/standard/StandardAsyncExecutionActor.scala
sed 's/\(\s*\)export _JAVA_OPTIONS."tmpDir\) "/\1\/tmp\.\$\$\$\$\ "/" \ backend/src/main/scala/cromwell/backend/standard/StandardAsyncExecutionActor.scala
sed 's/\(\s*\)export TMPDIR="tmpDir\) "/\1\/tmp\.\$\$\$\$\ "/" \ backend/src/main/scala/cromwell/backend/standard/StandardAsyncExecutionActor.scala
Edit backend/src/main/scala/cromwell/backend/standard/ StandardAsyncExecutionActor.scala
Replace line 380-381 with:
|mkdir -p $$tmpDir/tmp.$$$$
|export _JAVA_OPTIONS=-Djava.io.tmpdir="$$tmpDir/tmp.$$$$"
|export TMPDIR="$$tmpDir/tmp.$$$$
sbt clean
cp server/target/scala-2.12/cromwell-52-*-SNAP.jar ${GENOMICS_PATH}/cromwell/cromwell-52-fix.jar
```

 # 步骤 4

配置 Cromwell 工作环境，使用默认配置文件作为起点，并对其进行更改以满足实际场景需求。包括将 MariaDB 配置为数据库并添加 Slurm 后端用于工作流编排。

```
sed -i '$ a database {' reference.conf
sed -i '$ a \ \ profile = \"slick.jdbc.MySQLProfile$\"" reference.conf sed -i '$ a \ \ db {' reference.conf
sed -i '$ a \ \ \ \ driver = \"com.mysql.cj.jdbc.Driver\"" reference.conf
sed -i '$ a \ \ \ \ url = \"jdbc:mysql:\/\/localhost\/cromwell\?rewriteBatched\=true&serverTimezone=UTC\"" reference.conf
sed -i '$ a \ \ \ \ user = \"cromwell\"" reference.conf
sed -i '$ a \ \ \ \ password = \"cromwell\"" reference.conf sed -i '$ a \ \ \ \ connectionTimeout = 5000' reference.conf sed -i '$ a \ \ }' reference.conf
sed -i '$ a }' reference.conf
sed -i '479,484d' reference.conf
sed -i "479i \ \ \ \ \ \ \ \ job-id-regex\ =\ \"Submitted\ batch\ job\ (\ \ \ \ \ \ \ \d+).*\"" \ reference.conf
sed -i "479i \ \ \ \ \ \ \ \ check-alive\ =\ \"squeue\ -j\ \${job_id}\"" reference.conf sed -i "479i \ \ \ \ \ \ \ \ kill\ =\ \"scancel\ \${job_id}\"" reference.conf
sed -i "479i \ \ \ \ \ \ \ \ \"\"" reference.conf
sed -i "479i \ \ \ \ \ \ \ \ sbatch\ -J\ \${job_name}\ -D\ \${cwd}\ -o\ \${out}\ -e\ \${err}\ -t\ \${runtime_minutes}\ -p\ \${queue}\ \${"-c\ "+\ cpu}\ --mem\ \${memory_mb}\ --wrap\ \"/bin/bash\ \${script}\"" reference.conf
sed -i "479i \ \ \ \ \ \ \ \ submit\ =\ \"\"" reference.conf sed -i "479i \ \ \ \ \ \ \ \ \"\"" reference.conf
sed -i "479i \ \ \ \ \ \ \ \ String?\ docker" reference.conf
sed -i "479i \ \ \ \ \ \ \ \ String\ queue\ =\ \"haplo\"" reference.conf sed -i "479i \ \ \ \ \ \ \ \ Int\ memory_mb\ =\ 1024" reference.conf
sed -i "479i \ \ \ \ \ \ \ \ Int\ cpu\ =\ 2" reference.conf
sed -i "479i \ \ \ \ \ \ \ \ Int\ runtime_minutes\ =\ 600" reference.conf sed -i "479i \ \ \ \ \ \ \ \ runtime-attributes\ =\ \"\"" reference.conf sed -i "479i \ \ \ \ \ \ \ \ root\ =\ \"cromwell-slurm-exec\"" reference.conf sed -i "479i \ \ \ \ \ \ config\ {" reference.conf
sed -i "479i \ \ \ \ \ \ actor-factory\ =\ \"cromwell.backend.impl.sfs.config.\ ConfigBackendLifecycleActorFactory\"" reference.conf
sed -i "479i \ \ \ \ \ \ temporary-directory\ =\ \"$(/genomics_local/)\"" reference.conf sed -i "479i \ \ \ \ \ \ SLURM-HAPLO\ {" reference.conf
sed -i "479i \ \ \ \ \ \ }\ " reference.conf sed -i "479i \ \ \ \ \ \ }" reference.conf
sed -i "479i \ \ \ \ \ \ }\ " reference.conf
sed -i "479i \ \ \ \ \ \ \ \ job-id-regex\ =\ \"Submitted\ batch\ job\ (\ \ \ \ \ \ \ \d+).*\"" \ reference.conf
sed -i "479i \ \ \ \ \ \ \ \ check-alive\ =\ \"squeue\ -j\ \${job_id}\"" reference.conf
sed -i "479i \ \ \ \ \ \ \ \ kill\ =\ \"scancel\ \${job_id}\"" reference.conf sed -i "479i \ \ \ \ \ \ \ \ \"\"" reference.conf
sed -i "479i \ \ \ \ \ \ \ \ sbatch\ -J\ \${job_name}\ -D\ \${cwd}\ -o\ \${out}\ -e\ \${err}\ -t\ \${runtime_minutes}\ -p\ \${queue}\ \${"-c\ "+\ cpu}\ --mem\ \${memory_mb}\ --wrap\ \"/bin/bash\ \${script}\"" reference.conf
sed -i "479i \ \ \ \ \ \ \ \ submit\ =\ \"\"" reference.conf sed -i "479i \ \ \ \ \ \ \ \ \"\"" reference.conf
sed -i "479i \ \ \ \ \ \ \ \ String?\ docker" reference.conf
sed -i "479i \ \ \ \ \ \ \ \ String\ queue\ =\ \"bwa\"" reference.conf sed -i "479i \ \ \ \ \ \ \ \ Int\ memory_mb\ =\ 1024" reference.conf sed -i "479i \ \ \ \ \ \ \ \ Int\ cpu\ =\ 2" reference.conf
sed -i "479i \ \ \ \ \ \ \ \ Int\ runtime_minutes\ =\ 600" reference.conf sed -i "479i \ \ \ \ \ \ \ \ runtime-attributes\ =\ \"\"" reference.conf sed -i "479i \ \ \ \ \ \ \ \ root\ =\ \"cromwell-slurm-exec\"" reference.conf sed -i "479i \ \ \ \ \ \ config\ {" reference.conf
sed -i "479i \ \ \ \ \ \ actor-factory\ =\ \"cromwell.backend.impl.sfs.config.\ ConfigBackendLifecycleActorFactory\"" reference.conf
sed -i "479i \ \ \ \ \ \ temporary-directory\ =\ \"$(/genomics_local/)\"" reference.conf
sed -i "479i \ \ \ \ \ \ #\ Modifying\ temp\ directory\ to\ write\ to\ local\ disk" reference.conf sed -i "479i \ \ \ \ \ \ SLURM\ {" reference.conf
sed -i "479i \ \ \ \ providers\ {" reference.conf
sed -i "479i \ \ \ \ default\ =\ \"SLURM\"" reference.conf
```

步骤 5

通过测试，验证 Cromwell 是否配置正确。

```
su – Cromwell
cd ${GENOMICS_PATH}/cromwell
nohup java -jar -Dconfig.file=reference.conf cromwell-52-fix.jar server 2>&1 >>cromwell.log &
[cromwell@frontend cromwell]$ ps aux | grep cromwell | grep server
cromwell <PID> <..> java -jar -Dconfig.file=reference.conf cromwell-52-fix.jar server
cd ${GENOMICS_PATH}/cromwell/
cat >> HelloWorld.wdl << EOFwdl task hello {
String name command {
echo 'Hello ${name}!'
}
output {
File response = stdout()
}
runtime {
memory: "2MB" disk: "2MB"
}
}
workflow helloWorld { call hello
}
EOFwdl
echo {\"helloWorld.hello.name\":\ "World\"} >> HelloWorld.json
curl -v http://127.0.0.1:8000/api/workflows/v1 -
F \ workflowSource=@${GENOMICS_PATH}/cromwell/HelloWorld.wdl -
F \ workflowInputs=@${GENOMICS_PATH}/cromwell/HelloWorld.json
curl -v http://127.0.0.1:8000/api/workflows/v1/<id>/status
#After a few minutes, the above command should return a "succeeded" message.
```

步骤 6

下载并编译并安装 BWA（Burrows-Wheeler Aligner）。

```
su - cromwell
mkdir ${GENOMICS_PATH}/tools
wget https://github.com/lh3/bwa/releases/download/v0.7.17/bwa-0.7.17.tar.bz2
tar -xjf bwa-0.7.17.tar.bz2
cd bwa-0.7.17
cd ${GENOMICS_PATH}/tools
```

步骤 7

下载并安装 GATK，借助英特尔® AVX - 512 实现优化的英特尔® 基因组学内核库也将集成在其中。

```
cd ${GENOMICS_PATH}/tools
wget \
https://github.com/broadinstitute/gatk/releases/download/4.1.9.0/gatk-4.1.9.0.zip
unzip gatk-4.1.9.0.zip
echo "export PATH=\${GENOMICS_PATH}/tools/gatk-4.1.9.0:\$PATH" >> /etc/bashrc
cd ${GENOMICS_PATH}/tools
ln -s gatk-4.1.9.0 gatk
ln -s gatk-4.1.9.0/gatk-package-4.1.9.0-local.jar gatk-jar
```

步骤 8

下载并安装 Picard。

```
cd ${GENOMICS_PATH}/tools
wget https://github.com/broadinstitute/picard/releases/download/2.23.8/picard.jar
```

步骤 9

下载并安装 Samtools。

```
cd ${GENOMICS_PATH}/tools
wget https://github.com/samtools/samtools/releases/download/1.9/samtools-1.9.tar.bz2
tar -xjf samtools-1.9.tar.bz2
cd samtools-1.9
./configure -prefix=${GENOMICS_PATH}/tools
make
${GENOMICS_PATH}/tools
ln -s samtools-1.9 samtools
##### Summary of the commands
Here are all the commands in this section:
cd ${GENOMICS_PATH}/tools
wget https://github.com/samtools/samtools/releases/download/1.9/samtools-1.9.tar.bz2
tar -xjf samtools-1.9.tar.bz2
cd samtools-1.9
./configure -prefix=${GENOMICS_PATH}/tools
make
${GENOMICS_PATH}/tools
ln -s samtools-1.9 samtools
```

步骤 10

编译并安装 VerifyBamID2

```
cd ${GENOMICS_PATH}/tools
#https://github.com/broadinstitute/warp/tree/cec97750e3819fd88ba382534aaede8e05ec52df/dockers/broad/ VerifyBamId
cd $GENOMICS_PATH/tools
VERIFY_BAM_ID_COMMIT="c1cba76e979904eb69c31520a0d7f5be63c72253" GIT_HASH=$VERIFY_BAM_ID_COMMIT
HTS_INCLUDE_DIRS=$GENOMICS_PATH/tools/samtools-1.9/htslib-1.9/ HTS_LIBRARIES=$GENOMICS_PATH/tools/samtools-1.9/htslib-1.9/libhts.a
wget -nc https://github.com/Griffan/VerifyBamID/archive/$GIT_HASH.zip && \ unzip -o $GIT_HASH.zip && \
cd VerifyBamID-$GIT_HASH && \ mkdir build && \
cd build && \
echo cmake -DHTS_INCLUDE_DIRS=$HTS_INCLUDE_DIRS -DHTS_LIBRARIES=$HTS_LIBRARIES .. && \
CC=$(which gcc) CXX=$(which g++) \
cmake -DHTS_INCLUDE_DIRS=$HTS_INCLUDE_DIRS -DHTS_LIBRARIES=$HTS_LIBRARIES .. && \
make && \
make test && \ cd ../../
mv $GENOMICS_PATH/tools/VerifyBamID-$GIT_HASH $GENOMICS_PATH/tools/VerifyBamID && \ rm -rf $GENOMICS_PATH/tools/$GIT_HASH.zip $GENOMICS_PATH/tools/VerifyBamID-$GIT_HASH
sh ./build_verify.sh
```

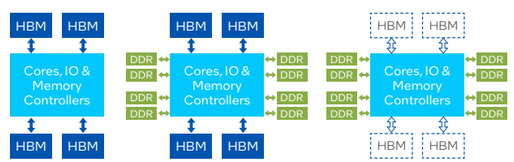
步骤 11

在最后，使用者还可以使用 20K Throughput Run 基准测试程序来确保上述软件栈中，针对计算集群的重要功能是否都已配置正确。

```
su - cromwell
cd ${GENOMICS_PATH}/cromwell
git clone https://github.com/Intel-HLS/BIGstack.git
vim configure
./step01_Configure_20k_Throughput-run.sh
./step02_Download_20k_Data_Throughput-run.sh
./step03_Cromwell_Run_20k_Throughput-run.sh
./step04_Cromwell_Monitor_Single_Sample_20k_Workflow.sh
step05_Single_Sample_20k_Workflow_Output.sh
./step05_Single_Sample_20k_Workflow_Output.sh
Total Elapsed Time for 64 workflows: 'X' minutes: 'Y' seconds
Average Elapsed Time for Mark Duplicates: 'X.YZ' minutes
```

部署在科学计算平台上的各类应用要获得更佳性能，除了需要强劲的算力以及经优化的并行计算效率之外，对内存带宽性能的关注同样不可或缺。一方面，内存带宽敏感型科学计算应用，如用于计算流体动力学（CFD）的 OpenFOAM 等，所需处理的网格数量动辄以亿计算。庞大的计算量和数据集会消耗大量内存带宽。另一方面，物理内核数量的不断增多，从某种意义上也“摊薄”了每个内核所能获得的内存带宽。

英特尔® 至强® CPU Max 系列处理器通过 HBM 内存的引入，来为这一问题的解决提供全新思路。如上一章节所述，如图 2-5-1 所示，英特尔® 至强® CPU Max 系列处理器为使用者提供了“仅 HBM”、“ HBM Flat”以及“HBM 缓存”三种内存模式，并根据计算集群的需要，配置为“Quadrant”和“SNC4”两种集群模式。



“仅 HBM”模式　“HBM Flat”模式　“HBM 缓存”模式

图 2-5-1 英特尔® 至强® CPU Max 系列处理器中 HBM 的三种内存模式

全局优化配置

以下配置，在三种不同 HBM 内存模式时均可参考使用（基于 Linux 环境）。

- 禁用内存交换。这项尤其适用于容量有限的“仅 HBM”模式。内存交换会严重影响性能。
- 启用 zone_reclaim_mode，其非常适合 NUMA 节点规模较小的场景（如“SNC4”集群模式）。启用该模式时，Linux 页面分配器会先在请求的 NUMA 节点上回收容易用的页面，然后再从其他 NUMA 节点上获得内存，从而减少不必要的 NUMA 交叉，避免性能下降。可使用如下命令开启：

```
1. echo 2 > /proc/sys/vm/zone_reclaim_mode
```

- 运行前，清理文件系统缓存，并使用以下命令规整内存

```
1. sync; echo 3 > /proc/sys/vm/drop_caches;
2. echo 1 > /proc/sys/vm/compact_memory
```

- 建议启用透明大页（Transparent Huge Pages，THP），大多数科学计算应用都会从 THP 的使用中获益。创建 THP 时可能会产生内存规整开销，但在每次运行前规整内存即可减少开销。
- 避免使用 /dev/shm (tmpfs) 来存储文件，从而减少可用内存。建议在启动作业前清除 /dev/shm。
- 建议使用最新的稳定版 Linux 内核。

“仅 HBM”与“HBM 缓存”内存模式下优化配置

对于使用“仅 HBM”和“HBM 缓存”两种内存模式的场景，使用者无需额外的配置操作。但由于 HBM 容量有限，可采取一些额外操作来减少内存容量开销，例如：

- 减少系统启动时的非必要服务（守护进程）和驱动程序（如打印或邮件服务等守护进程等）；
- 缩减操作系统文件缓存容量和 MPI 缓冲区；
- 使用时建议在 OpenMP 线程数与 MPI 等级之间取得平衡。由于 OpenMP 线程共享内存，因此多使用 OpenMP 线程可以减少内存总占用空间；
- 如应用无法适配 HBM 容量，需要适当调整 OpenMP 堆栈大小和 MPI 通信缓冲区大小；
- 确保没有 NUMA 失中（通过在运行程序前后运行 numastat 来实现）；
- 如上述步骤仍无法应用使适配 HBM 容量，可考虑扩展到更多节点。

在集群模式下，使用者可以在 BIOS 中进行集群模式的修改，以英特尔白牌机为例，设置 SNC4 集群模式的路径为：[Advanced] -> [Memory Configuration] -> [Memory RAS and Performance Configuration] -> SNC，设置成 SNC off 即是 Quadrant 模式。对于 OEM 的服务器，该选项一般在另一个位置，例如 [Uncore Configuration] -> [Uncore General Configuration] -> SNC (Sub NUMA)，使用者需要根据对应服务器机型的说明进行配置。

同时，在“Quadrant”和“SNC4”集群模式设置完成后，使用者可使用 numactl -H 验证 NUMA 节点配置和每个 NUMA 节点的总内存或可用内存容量。

“HBM Flat”内存模式下优化配置

在完成 DDR 内存安装后，首先需要在 BIOS 设置中选择 1LM 以启用“HBM Flat”模式。以英特尔白牌机为例，设置 1LM 的路径为：[Advanced] -> [Memory Configuration] -> HBM mode = 1LM。对于 OEM 的服务器，该选项一般在另一个位置，例如 [Socket Configuration] -> [Memory Configurations] -> [Memory Map] ->[Volatile Memory Mode] ->1LM，使用者需要根据对应服务器机型的说明进行配置。但此时 HBM 因为被标记为专用内存，仅有 DDR 内存对操作系统和应用可见，需要采取额外的操作步骤才能使 HBM 进入默认内存池。

 # 步骤 1

安装以下 Linux 包：

```
1. dnf install daxctl ndctl
```

 # 步骤 2

针对双路系统执行下列 daxctl 命令（其中“Quadrant”模式仅需前两项命令，“SNC4”模式要求完成所有命令）。所有这些命令均需 root 权限。

```
1. ## Base commands for both Quadrant andSNC4 cluster modes
2. ##
3. daxctl reconfigure-device -m system-ram dax0.0
4. daxctl reconfigure-device -m system-ram dax1.0
5. ## For SNC4 cluster mode, use the following additional commands:
6. ##
7. daxctl reconfigure-device -m system-ram dax2.0
8. daxctl reconfigure-device -m system-ram dax3.0
9. daxctl reconfigure-device -m system-ram dax4.0
10. daxctl reconfigure-device -m system-ram dax5.0
11. daxctl reconfigure-device -m system-ram dax6.0
12. daxctl reconfigure-device -m system-ram dax7.0
```

每次启动系统时，都需要执行步骤 2，建议将命令写入脚本并在操作系统启动时自动执行。完成后可使用 numactl -H 验证 HBM 节点是否可见，以及整个 HBM 容量是否可用。

在“HBM Flat”模式下，对应用进行 HBM 布局非常重要，使用者可使用标准的 Linux 工具 numactl 来进行该操作，也可以使用英特尔® MPI。以使用英特尔® MPI 为例，环境变量 I_MPI_HBW_POLICY 可以针对 MPI 等级分配 HBM。

```
1. mpirun -genv I_MPI_HBW_POLICY hbw_bind -n 2 ./a.out
2. mpirun –genv I_MPI_HBW_POLICY hbw_preferred -n 2 ./a.out
3. mpirun –genv I_MPI_HBW_POLICY hbw_interleave -n 2 ./a.out
```

I_MPI_HBW_POLICY 环境变量也接受由 MPI 本身的内存分配策略。例如以下命令在进行用户和 MPI 库分配时都使用了 hbw_bind 策略。

```
1. mpirun -genv I_MPI_HBW_POLICY hbw_bind,hbw_bind -n 2 ./a.out
```

值得注意的是，在 RHEL（Red Hat Enterprise Linux）Linux 的 8.x 版本中，不支持通过以上方式激活 HBM，使用者需要在在启动时，在 kernel 选项中增加 efi=nosoftreserve。

基准测试
实战篇

得益于芯片技术的推陈出新、软件优化工具和优化方法的不断完善，各类科学计算平台的性能在近年来已获得令人赞叹的提升与突破。在这一过程中，借助各类基准测试（Benchmark）程序或脚本对科学计算平台的性能进行综合测试与评估，也成为不可或缺。通过基准测试，用户可以对平台的处理性能和工作效率有着更加清晰的认知，从而可通过在不同的应用场景中部署更为适宜的产品，在提升效能的同时也有效降低成本。

高性能 Linpack（High Performance Linpack，HPL）和高性能共轭梯度（High Performance Conjugate Gradient，HPCG）是目前运用广泛的科学计算平台基准测试，而 Stream 是目前广为使用的内存带宽性能基准测试。为了让用户能更好地对基于英特尔® 架构的科学计算平台做出较为精准的性能预期，并由此制定更合理的科学计算方案，英特尔也在标准版 HPL、HPCG 和 Stream 的基础上，推出了经优化和性能增强的版本。

Linpack 基准性能测试英特尔® 发行版

Linpack 基准性能测试英特尔® 发行版简介

线性系统软件包（Linear system package，Linpack）是一种用于计算机浮点性能测试的基准测试，其核心方法是使用高斯消元法（Gaussian Elimination）求解 N 元一次稠密线性代数方程组的测试，由此评价计算机系统的浮点性能。Linpack 测试包括三类，Linpack 100、Linpack 1000 和 HPL。其中 Linpack 100 是求解规模为 100 阶的稠密线性代数方程组，Linpack 1000 是求解 1000 阶的线性代数方程组。

HPL 是面向科学计算系统（例如基于科学计算平台的解决方案）提出 Linpack 扩展测试。用户可在不修改测试程序的基础上，调节矩阵大小以及所使用算力规模等参数，并可使用各种优化方法来执行测试以获取最佳性能，从而对高性能的科学计算平台做出准确评估。HPL 目前也是 TOP 500（http://www.top500.org）所使用的基准测试之一。

更多 HPL 信息，请参阅：http://www.netlib.org/benchmark/hpl/。

Linpack 基准性能测试英特尔® 发行版（以下简称 “Linpack 英特尔® 发行版”）是在标准 HPL 的基础上加入了众多优化和增强功能（例如使用英特尔® MPI）的新版本，其同样也可用于 TOP500 测试。与标准 HPL 测试中每个 MPI 进程需要在类似的 CPU 和内存环境中运行不同，Linpack 英特尔® 发行版对异构平台有着良好的支持，只要计算节点有着足够的算力和内存资源，数据就可以根据每个节点的性能要求进行平衡。英特尔® oneMKL 提供的预置二进制文件可静态或动态链接到英特尔® MPI 库。同时，使用者也可使用英特尔® oneMKL 内置的 MPI 包装器，创建一个与自定义 MPI 之间链接的二进制文件。

LIinpack 英特尔® 发行版包括了与英特尔® MPI 库链接的预构建二进制文件，以及在自定义 MPI 场景下构建二进制文件的工具（基于英特尔® oneMKL 内置的 MPI 包装器）。所有文件都位于英特尔® oneMKL 所在目录的子目录 ./benchmarks/mp_linpack/ 中。

配置 Linpack 英特尔® 发行版

在英特尔® oneAPI HPC 工具包（详细介绍请参阅产品技术篇）中，包含了已编译的 mp_linpack，使用者可以直接使用。

配置参数

HPL.dat 中最重要的参数是 P、Q、NB 和 N。可按如下方式进行配置。

- P 和 Q：分别是进程网格的行数和列数，P*Q 必须是 HPL 正在使用的 MPI 进程的数量，P ≤ Q，并且 P*Q 尽量接近正方形。
- NB：数据分布的块数量，可根据英特尔® 处理器平台类型进行设置，参考值详见：https://www.intel.com/content/www/us/en/docs/onemkl/developer-guide-linux/2023-0/configuring-parameters.html
- N（problem size）：对于同计算平台运行的场景，选择可被 NB*LCM（P, Q）整除的 N，其中 LCM 是这两个数字的最小公倍数。对于异构计算平台运行的场景，可参阅 Linpack 英特尔® 发行版相关信息，了解如何选择 N 值。

便捷的命令行参数

Linpack 英特尔® 发行版对 HPL 命令行参数提供了支持，这些参数可以帮助使用者更加灵活的更改配置而无需频频修改 HPL.dat 文件。命令行中的 <> 表明其为可修改参数。

```
1. ./xhpl -n <problem size> -m <memory size in Mbytes> -b <block size> -p <grid row dimn> -
   q <grid column dimn>
```

使用者也可将命令行参数与示例 runme 脚本一起使用。

```
1. ./runme_intel64_dynamic -m <memory size in Mbytes> -b <block size> -p <grid row dimn> -
   q <grid column dimn>
```

- 举例如下，在 HPL.dat 和脚本中的其他参数正确的情况下，可通过以下命令，在 1x3 网格上运行 N=10000：

```
1. ./runme_intel64_dynamic -n 10000 -p 1 -q 3
```

- 通过 m 参数，使用者可根据内存大小而不是 N（problem size）进行缩放。m 参数仅指矩阵存储的大小。因此要在 16 个节点上使用 NB=256 的 50000 兆字节的矩阵，需要通过调整脚本来将 MPI 进程的总数设置为 16，并执行以下命令：

```
1. ./runme_intel64_dynamic -m 50000 -b 256 -p 4 -q 4
```

运行 Linpack 英特尔® 发行版

基于自定义 MPI 场景编译 Linpack 英特尔® 发行版

在具有多个 MPI 进程的单节点，或多个节点上运行 Linpack 英特尔® 发行版，使用者需要使用 MPI 并修改 HPL.dat，或使用命令行参数。以下示例将简述如何运行使用脚本动态链接的预置 Linpack 英特尔® 发行版二进制文件。若运行其他二进制文件，请相应调整步骤。

 # 步骤 1

为英特尔® MPI 库和英特尔® 编译器加载必要的环境变量。

```
1. <compiler directory>/env/vars.sh
2. <mpi directory>/env/vars.sh
```

 # 步骤 2

在 HPL.dat 中，将 N（problem size）设置为 10000。为获得更好性能，在系统上启用 NUMA，并在每个 NUMA 套接字上配置运行一个 MPI 进程，如下所述。

- 参阅 BIOS 设置，在系统上启用 NUMA；
- 根据集群配置，在 runme_intel64_dynamic 脚本的顶部设置以下变量：MPI_PROC_NUM（MPI 进程的总数）、MPI_PROC_NUM（每个集群节点的 MPI 进程数）；
- 在 HPL.dat 文件中，设置参数 Ps 和 Qs，使 Ps*Qs 等于 MPI 进程的数量。例如，对于 2 个进程，将 Ps 设置为 1，将 Qs 设置为 2。或保持 HPL.dat 文件的原样，但使用 -p 和 -q 命令行参数启动。

 # 步骤 3

运行 runme_intel64_dynamic 脚本。

```
1. ./runme_intel64_dynamic
```

 # 步骤 4

重新运行测试，增加 N（problem size）直到矩阵使用了约 80% 的可用内存。要执行此操作，请修改 HPL.dat 第 6 行中的 Ns，或者使用 -n 命令行参数。例如对于 32GB 内存，Ns 设为 56000，对于 64GB 内存，Ns 设为 83000。

Linpack 英特尔® 发行版对异构计算平台的支持

Linpack 英特尔® 发行版对异构计算平台同样有着良好的支持。其核心思想是基于特定的相对性能，在每个节点上进行负载均衡，即在较弱节点上使用更少的内存，达到 “能力越大、责任越大” 的效果。使用者首先可以在 MPI 节点文件中，按照性能从高到低对节点名进行排列，然后通过调整 HPL.dat 文件来指定异构因子的数量以及每一个异构因子。并指定不同类型处理器节点的列数以及存在的限制。异构因子决定了在不同节点上的强度。例如设为 2.5 时，那么更强大的节点上大约有 2.5 倍的强度。具体步骤如下：

 # 步骤 1

测试每一个节点的单节点 HPL 性能，每个节点 1 个 MPI 进程，然后在 nodefile 中将节点名按照性能降序排列。

 # 步骤 2

在多节点上运行 HPL，获得 HPL.dat 参数的最优值，例如 P x Q。

```
1    PMAP process mapping (0=Row-,1=Column-major)
```

 # 步骤 3

在 HPL.dat 的最后加入：

```
2            number of heterogeneous factors    #架构（型号）种类的数量
0 1 2.7      [start_column, stop_column, heterogeneous factor for that range]
```

 # 后续每一行代表一种架构（型号），第一个数字是起始的列号，第二个数字是终止的列号，第三个数字是异构因子（不同架构的内存占用比例，或性能比例）

……

 # 步骤 4

每一种架构（型号）的节点数必须是 P 的倍数，最多可以有 Q 种不同的架构（型号），每一种架构异构因子 *Fi* 由第一步当中实测的性能值得来。如图 3-1-1 所示，估算 N（problem size）：假设有 *n* 种架构（型号），每种架构（型号）有 *Ci* 列，每种架构（型号）的内存大小是 Mi 字节，每种架构（型号）的异构因数是 *Fi*。

$$N = \sqrt{\frac{\text{内存占用率} \times M_{min} \times P \times \sum_{i=1}^{n}(F_i \times C_i)}{8 \times F_1}}, \quad \text{其中 } M_{min} = min\frac{F_i}{F_1}M_i, \text{ 内存占用率可以取 90\%}$$

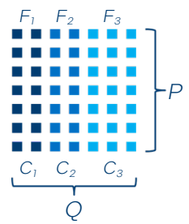


图 3-1-1 在异构节点上运行 Linpack 英特尔® 发行版

Linpack 英特尔® 发行版可通过以下命令在异构平台上设置工作量。命令行参数包括：n 用于描述 problem size、f（异构因子）用于控制分配给不同节点的工作量、c 控制节点上的进程列数等，例如：

```
1.  ./xhpl –n <problem size> –b <block size> –p <grid row dimn> –q <grid column dimn> –
    f <heterogeneous factor> –c <number of faster processor columns>
```

由于在同构节点上，英特尔® 睿频加速技术的使用也会导致不同节点之间性能偏差，传统方法使得系统的总性能受性能最低的节点限制（“木桶原理”），因此在实战中也可在同构节点上使用异构模式运行 Linpack 英特尔® 发行版。

更多 Linpack 英特尔® 发行版在异构计算平台上的配置方法，请参阅：
https://www.intel.com/content/www/us/en/docs/onemkl/developer-guide-linux/2023-0/support-in-the-intel-disti-for-linpack-benchmark.html

提升运行性能

使用者可按照以下步骤来提升运行性能：

 # 步骤 1

重启所有节点；

 # 步骤 2

确保所有节点都处于相同的条件下，且之前的 HPL 运行没有留下任何僵尸进程。在每个节点上运行单节点 Stream（一种内存带宽性能测试工具）和 Linpack 英特尔® 发行版。确保各个节点的结果差在 10% 以内（一般而言，problem size 越大，测试结果越好。但内存需求和运行时间也会增加，其中内存占用量为平方关系增加，运行时间为立方关系增加。为取得最好结果，内存占用量可达到 95%；为快速得到测试结果，可以按照按照每路处理器占用 64GB 内存计算 problem size。）；

 # 步骤 3

检查集群互连是否正常工作。使用针对带宽和延迟的 MPI 测试（如英特尔® MPI 基准测试包中的测试）对整个集群上运行测试；

 # 步骤 4

在两个或四个节点对上运行 Linpack 英特尔® 发行版，确保结果相差不超过 10%。（problem size 必须足够大，具体取决于内存大小和处理器速度）；

 # 步骤 5

在整个集群上运行一个小 problem size 的用例以确保正确性；

 # 步骤 6

增加 problem size 并运行实际测试负载；

 # 步骤 7

如果出现问题，返回步骤 2。

在异构计算平台进行运行之前，先在同计算平台上运行等效程序。

面向英特尔® 至强® CPU Max 系列处理器的运行

Linpack 英特尔® 发行版可在单节点或多节点的英特尔® 至强® CPU Max 系列处理器平台上运行。例如，要在采用双路系统的单节点上以 “仅 HBM” 模式和 “SNC4” 集群模式运行，须更改 runme_intel64_dynamic 文件中的以下定义：

```
1.  export MPI_PROC_NUM=2
2.  export MPI_PER_NODE=2
3.  export NUMA_PER_MPI=4
```

然后运行以下命令：

```
1.  ./runme_intel64_dynamic -p 2 -q 1 -b 384 -n 120000
```

面向英特尔® 架构优化的 HPCG 基准测试

作为另一种面向科学计算系统性能测试的重要方法，HPCG 基准测试的核心是通过预处理共轭梯度（preconditioned conjugate gradient, PCG）算法来求解稀疏矩阵方程组。HPCG 基准测试是面向一个 3D 规则区域上已经 27 点离散化的椭圆偏微分方程，在所有的可用 MPI 列中，调用一个 3D 域去填充 3D 虚拟过程网格的过程。HPCG 结合了局部和对称高斯 - 塞德尔（Gauss-Seidel）预处理步骤（该步骤需要三角形前向求解和反向求解），使用预处理共轭梯度算法来求解中间方程组。

在上述每个预处理步骤中，会使用合成的多网格 V 循环从而使基准测试更适合实际应用。HPCG 局部实现矩阵乘法，在相邻进程之间进行初始 Halo 数据交换。与 Linpack 系列基准测试关注线性方程的计算性能不同，HPCG 使用更为复杂的微分方程计算方式，这就使其更为贴近一些实际场景，与许多科学工作负载相似。HPCG 目前也是 TOP500 所使用的基准测试之一。更多 HPCG 信息，请参阅：http://hpcg-benchmark.org。

面向英特尔® 架构优化的 HPCG 基准测试简介

面向英特尔® 架构优化的 HPCG 基准测试（以下简称“英特尔® 优化版 HPCG”）在标准 HPCG 的基础上，针对英特尔® 架构的平台以及英特尔® AVX - 512 等技术的特性进行了优化和功能增强，旨在提供更有效的高性能计算系统评估方法力，为用户在设计和部署科学计算平台方案时提供更好的性能参考和效果评估能力。

英特尔® 优化版 HPCG 是基于标准版 HPCG v3.0 参考实现的源代码，并加入了以下必要的修改和增强。

■ **面向英特尔® 架构的优化**

■ **链接到英特尔® oneMKL 的预编译基准测试可执行文件，包括：**

- 用于 SpMV（Sparse Matrix-Vector Multiplication，稀疏矩阵向量乘法）的 Inspector-executor 稀疏 BLAS（Basic Linear Algebra Subprograms，基础线性代数子程序库）核心；
- TRSV（Sparse triangular solve，稀疏三角形解）；
- SYMGS（Symmetric Gauss-Seidel smoother，对称高斯 - 塞德尔平滑器）。

在英特尔® 优化版 HPCG 的软件包中，包含了适用于英特尔® MPI 5.1 或更高版本的预编译 HPCG 基准测试。基准测试的所有文件都位于英特尔® oneMKL 所在目录的 ./benchmarks/hpcg nchmarks/hpcg 子目录中。针对不同处理器平台，使用者可选择不同的版本，如 xhpcg_knl、xhpcg_skx 等。英特尔® 优化版 HPCG 的软件包还包含了为其它 MPI 实现（如 Open MPI: 英特尔® AVX - 512 优化版）编译基准测试所需的源代码。hpcg.dat 是基准测试的输入文件。

使用面向英特尔® 架构优化的 HPCG 基准测试

可以按照以下步骤来开始使用英特尔® 优化版 HPCG。

 # 步骤 1

在集群文件系统上，将英特尔® 优化版 HPCG 软件包解压缩到所有节点都可访问的目录中，阅读软件包中的 readme.txt 文件中的说明并接受许可证。

 # 步骤 2

将目录切换为 hpcg/bin。

 # 步骤 3

根据硬件平台的类型，确定最适合使用的基准测试的预编译版本。

 # 步骤 4

确保英特尔® oneMKL、英特尔® 编译器以及 MPI 运行时的环境设置正确。使用者可使用这些发行版中包含的 scriptsvars.sh、compilervars.sh 和 mpivars.sh 来完成设置。

 # 步骤 5

运行基准测试。在处理器内置英特尔® AVX - 512（或英特尔® AVX、英特尔® AVX 2）技术时，为每一路处理器设置一个 MPI 进程，每个内核设置一个 OpenMPI 线程，且跳过同步多线程（simultaneous multithreading，SMT）设置。

设置 KMP_AFFINITY=granularity=fine,compact,1,0。在一个具有 128 节点的计算集群上，每个节点部署有双路英特尔® 至强® 铂金 8452Y 处理器，可运行以下命令：

```
1. mpiexec.hydra -genv KMP_AFFINITY granularity=fine,compact,1,0 -genv OMP_NUM_THREADS 36 -n 256 -ppn 2 ./bin/xhpcg)_avx512 -n192 -t60#
```

 # 步骤 6

当基准测试完成执行（通常需要几分钟）后，可在当前目录中找到包含结果的 YAML 文件。文件一般显示为：

HPCG result is VALID with a GFLOP/s rating of: [GFLOP/s]

选择最佳参数配置和 Problem Sizes

英特尔® 优化版 HPCG 的性能取决于许多系统参数，例如系统的硬件配置和所使用的 MPI 实现。要在特定环境下获得最佳性能，请选择以下参数的组合：

- 每个主机的 MPI 进程数和每个进程的 OpenMPI 线程数；
- 本地 problem size。

在基于英特尔® 架构的处理器集群上，根据所支持的英特尔® AVX - 512 等指令集，每路处理器运行一个 MPI 进程，每个物理处理器内核运行一个 OpenMP 线程，且跳过 SMT 线程。

方案中可以使用足够大的 problem size 来获得更优性能，以便更好地利用可用的处理器内核。但 problem size 也不宜过大，以保证所有任务都适合可用的内存。

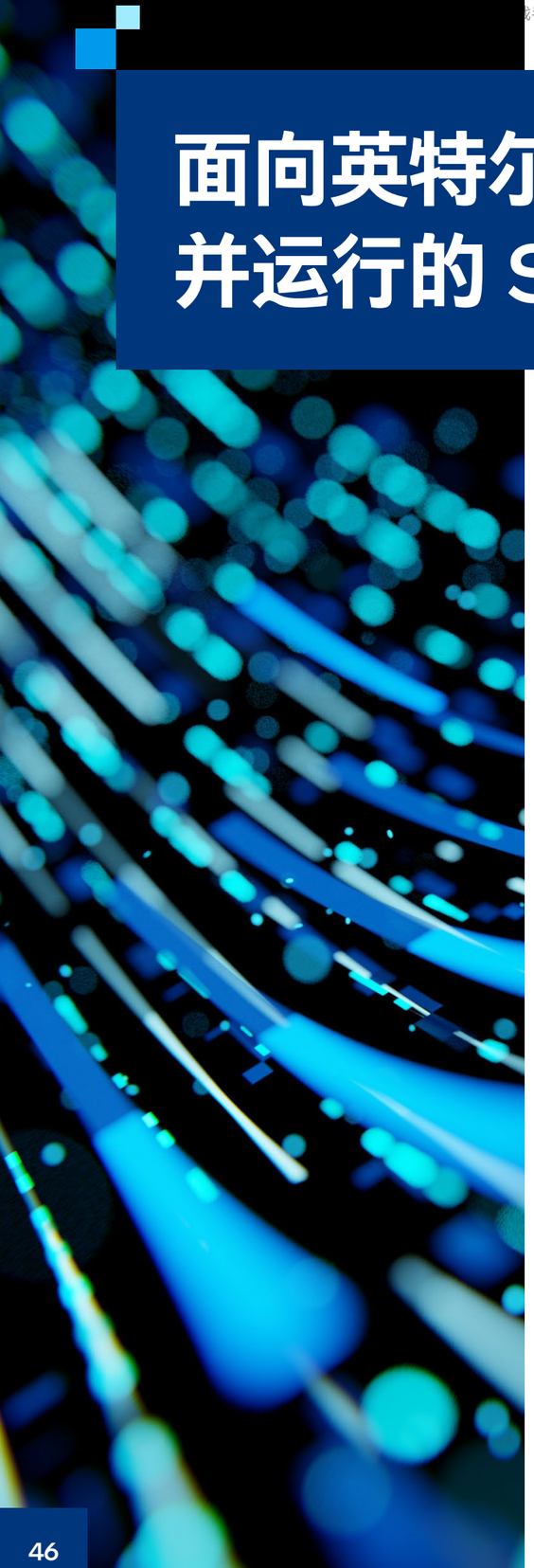
面向英特尔® 至强® CPU Max 系列处理器的运行

面向英特尔® 至强® CPU Max 系列处理器使用英特尔® 优化版 HPCG，可先使用以下命令行从源代码进行编译：

```
1.  # source C/C++ compiler, MPI compiler, and MKL library
2.  #
3.  export MKLROOT=/path/to/mkl
4.  export LD_LIBRARY_PATH=${MKLROOT}/lib/intel64:${LD_LIBRARY_PATH}
5.  # build binary for Intel AVX-512 -- bin/xhpcg_skx will be created
6.  #
7.  ./configure IMPI_IOMP_SKX
8.  make -j4 MKLROOT=${MKLROOT} MKL_INCLUDE=${MKL
```

假设以“仅 HBM”内存模式和“SNC4”集群模式在采用英特尔® 至强® CPU Max 系列处理器（每个处理器有 56 个内核）的双路系统上运行，可使用以下命令行：

```
1.  # Note: Select the best MPI x OMP decomposition for your case
2.  # Following assumes SNC4 (8 NUMA nodes on 2S),
3.  # and 14 cores (28 threads) on a NUMA node
4.  #
5.  export MKL_NUM_THREADS=28
6.  export OMP_NUM_THREADS=28
7.  nprocs_per_node=8
8.  nnodes=1
9.  nprocs=$((nnodes*nprocs_per_node))
10. problem_size=168 # options: 168, 192, 256
11. run_time_in_seconds=100 # 100 used as smoke test.
12. # 1800 is min for official HPCG submission
13. export I_MPI_SHM=spr-hbm
14. export I_MPI_FABRICS=shm:ofi
15. export I_MPI_PIN_DOMAIN=numa
16. export I_MPI_DEBUG=10 # print out mpi configuration mapping data
17. # for 1 hyper-thread, use 'compact,1,0' instead of 'compact'
18. #
19. export KMP_AFFINITY=granularity=fine,compact
20. echo " ==="
21. echo " ==="
22. echo " ==="
23. nnodes: ${nnodes}"
24. ppn: ${nprocs_per_node}"
25. nprocs: ${nprocs}"
26. echo " === n_omp_per_proc: ${MKL_NUM_THREADS}"
27. echo " === prob_size: ${problem_size}"
28. echo " === run_time: ${run_time_in_seconds}"
29. # run bin/xhpcg_skx binary (either prebuilt or built by user)
30. #
31. mpiexec.hydra -genvall -n ${nprocs} -ppn ${nprocs_per_node} bin/xhpcg_skx -
32. n$problem_size -t$run_time_in_seconds
```



面向英特尔® 架构平台编译并运行的 Stream 基准测试

部署在科学计算平台中的许多应用都属于内存带宽性能敏感型，因此选择适宜的内存带宽性能基准测试同样必不可少。Stream 是目前广为使用的内存带宽性能基准测试，其通过数组的复制（Copy）、数组的尺度变换（Scale）、数组的矢量求和（Add）以及数组的复合矢量求和（Triad）这四种数组的运算来执行测试。一般来说，Stream 测试中使用的数组大小，至少应为处理器所有末级缓存总和的 4 倍，或 100 万个数，取其中的较大者。

下文提供了面向英特尔® 架构处理器平台的 Stream 基准测试的编译和运行参考方法，以此帮助科学计算平台的使用者在其上获得理想的性能。Stream 有两种使用类型，其中 Standard 是从未修改的源代码中获得的测试结果，Tuned 是经使用者或供应商修改源代码后获得的测试结果。在基于英特尔® 架构的处理器平台上，使用者不需要任何修改就能够获得最佳结果，因此基于英特尔® 架构处理器平台的 Stream 属于 Standard 类型。可以在以下网页下载包括 Stream 源代码以及编译执行脚本：https://github.com/intel/memory-bandwidth-benchmarks。

编译前准备

- **英特尔® 编译器**：Stream 基准测试的性能取决于所使用编译器的选项。本文方法中使用英特尔® 编译器生成底层非临时存储指令，以在英特尔® 架构处理器平台上实现更佳性能。
- **Linux 环境**：目前 makefile 采用的是 Linux 操作系统环境。

编译 Stream 基准测试

确认英特尔® 编译器可用；

- 运行 make 命令，例如处理器支持英特尔® AVX - 512，得到 stream_avx512.bin（同样，英特尔® AVX、英特尔® AVX 2 也可以获得相应二进制文件）；

- 默认情况下，以下 Stream 配置参数用于编译二进制文件：

```
1. STREAM_TYPE = double
2. STREAM_ARRAY_SIZE = 269000000 (this translates to about 2 GB per array of memory footprint)
3. NTIMES = 100
4. OFFSET = 0
```

- Makefile 支持如下选项：

  size=<number_of_elements_per_array>

  cpu=<avx,avx2,avx512>

  rfo=1 forces to use regular cached stores instead of non-temporal stores

  help

参考示例如下：
仅为支持英特尔® AVX - 512 的处理器平台编译 Stream 基准测试，执行：

```
1. make CPU=AVX512
```

为每个缓冲区包含 67,108,864 个元素，支持英特尔® AVX - 512 的处理器平台编译 Stream 基准测试，执行：

```
1. make size=671008864 CPU=AVX512
```

显式使用常规缓存存储，执行：

```
1. make rfo=1
```

运行 Stream 基准测试

使用者可通过一个基准测试运行脚本（run.sh）来运行 Stream 基准测试，其中：

- **二进制文件**：使用编译步骤中产生的最合适 Stream 二进制文件，例如处理器支持英特尔® AVX - 512，就选择 stream_avx512.bin；
- **OpenMP 设置**：将 OMP_NUM_THREADS 设置为系统上的物理内核数量。KMP_AFFINITY 设置为 compact pinning。即便系统启用的超线程设置也请忽略超线程内核；
- 将结果存储到日志文件中。此外输出相关的系统信息，如套接字、内核、线程、NUMA 域、内存子系统等的数量。在解析 dmidecode 的输出时，使用 sudo 运行会获得内存子系统的更详细信息。

面向英特尔® 至强® CPU Max 系列处理器的运行

为了在英特尔® 至强® CPU Max 系列处理器上实现更出色的性能，首先可以使用以下命令行启用软件预取：

```
1. icc-O3-xCORE-AVX512-qopt-zmm-usage=high-mcmodel=large-qopenmp-qopt-streaming-stores=always-fno-builtin-qopt-prefetch-distance=128,16-DSTREAM_ARRAY_SIZE=500000000-DNTIMES=500 stream.c-o stream
```

使用以下命令执行生成的二进制文件：

```
1. KMP_HW_SUBSET=lt KMP_AFFINITY=balanced,granularity=core,verbose ./stream
```

产品
技术篇

第四代英特尔® 至强® 可扩展处理器

第四代英特尔® 至强® 可扩展处理器旨在为人工智能、数据分析、存储和科学计算方面快速增长的工作负载提供性能加速。该处理器具备多种内置加速器，帮助客户将零信任安全策略付诸实践，同时利用先进的安全技术，即使面对敏感或受监管的数据，也能解锁新的商业合作机会和洞察。使用这款处理器可跨多个云和边缘环境进行扩展，满足自身的部署需求。英特尔® 至强® 可扩展处理器具有很强的灵活性，可在其上选择不同的云服务，帮助企业顺利实现应用移植。

基础性能进一步大幅提升

- 第四代英特尔® 至强® 可扩展处理器采用全新架构，单核性能比上一代产品更高，每路配备多达 60 个内核。每个系统支持单路、双路、四路或八路配置。为了与内核数增加这种情况相匹配，该平台在内存和 I/O 子系统方面也做了相应改进。DDR5 内存提供的带宽和速度与 DDR4 相比提高多达 1.5 倍，速率达到 4,800 MT/s[6]。此外，该平台还具有每路 80 条 PCIe Gen5 通道的特点，与之前的平台相比，I/O 得到显著提升。本代处理器还可提供 CXL 1.1 连接，支持高网络带宽并使附加加速器能够高效运行。第四代英特尔® 至强® 可扩展处理器支持的技术支持根据工作负载要求的变化灵活扩展和调整。此外，本代处理器还可助力实现以下优势：
- 进一步提升网络、存储和计算性能，并通过将繁重的任务卸载到英特尔® 基础设施处理单元（Intel® Infrastructure Processing Unit，英特尔® IPU）来提高 CPU 利用率；
- 通过英特尔® UPI 2.0 提高多路带宽（高达 16 GT/s）；
- 使用英特尔® Speed Select 技术（英特尔® SST）调整 CPU 配置，满足特定工作负载的需求；
- 增加三级缓存（LLC）共享容量（所有内核共享多达 100 MB LLC）；
- 通过硬件增强型安全功能加强对安全态势的掌控；
- 使用英特尔® Virtual RAID on CPU（英特尔® VROC），从而无需再用单独的 RAID 卡。

内置众多加速引擎，重新定义性能

与增加 CPU 内核数相比，内置加速器是一种提升性能更有效的方法。其不但可以提高 CPU 利用率，降低功耗，并提高投资回报率（ROI），同时还能帮助企业实现可持续发展目标。英特尔® 至强® 可扩展处理器支持广泛且独特的内置加速器，有助于提高性能和效率，减少另行添置专用硬件的需求。在云端和本地环境中，这些专用功能支持人工智能、安全性、科学计算、数据分析、存储和网络等目前最为常见的严苛工作负载。

- **科学计算**：第四代英特尔® 至强® 可扩展处理器可以提高科学计算工作负载中常见的多线程代码性能。这些工作负载包括制造业仿真、分子动力学、地球系统建模以及 AI 推理和训练。该处理器内置的加速器可提供较高的精度水平，同时还能加速多种 AI 数据类型的处理。该处理器还支持 DDR5 内存、PCIe Gen5、英特尔® 超级通道互联（Intel® Ultra Path Interconnect，英特尔® UPI）2.0 和 Compute Express Link（CXL），显著提升了整体数据的吞吐量。
- **AI**：凭借更优的矢量指令和矩阵乘法运算，第四代英特尔® 至强® 可扩展 处理器展现出更为出色的 AI 推理和训练性能。英特尔® AMX 可以显著提高推荐系统、NLP、图像识别、媒体处理和分发以及媒体分析等深度学习工作负载的性能。

第四代英特尔® 至强® 可扩展处理器的新特性或新功能

PCI Express Gen5（PCIe 5.0）

带来全新的 I/O 速度，可在 CPU 和互联设备之间实现更高的吞吐量。第四代英特尔® 至强® 可扩展处理器具有多达 80 条 PCIe 5.0 通道，非常适合高速网络、高带宽加速器和高性能存储设备。PCIe 5.0 的 I/O 带宽是 PCIe 4.0 的两倍，仍具备向后兼容性并提供用于 CXL 连接的基础插槽[7]。

DDR5

以更高内存带宽克服数据瓶颈，提高计算性能。与 DDR4 相比，DDR5 的带宽提高多达 1.5 倍，因此有机会提升性能、容量和能效并降低成本[8]。借助 DDR5，第四代英特尔® 至强® 可扩展处理器提供的速率可高达 4,800 MT/s（1 DPC）或 4,400 MT/s（2 DPC）。

CXL

借助面向下一代工作负载的 CXL 1.1，降低数据中心的计算时延并帮助减少 TCO。CXL 是另一种跨标准 PCIe 物理层运行的协议，可以在同一链路上同时支持标准 PCIe 设备和 CXL 设备。CXL 可带来的一大关键能力是在 CPU 和加速器之间创建统一且一致的内存空间，它将革新未来数年数据中心服务器架构的构建方式。

6、7、8 https://www.intel.cn/content/www/cn/zh/products/docs/processors/xeon-accelerated/4th-gen-xeon-scalable-processors-product-brief.html

- **英特尔® 高级矩阵扩展（Intel® Advanced Matrix Extensions，英特尔® AMX）**可加速自然语言处理（NLP）、推荐系统和图像识别等深度学习（DL）推理和训练工作负载。
- **英特尔® 数据流加速器（Intel® Data Streaming Accelerator，英特尔® DSA）**可通过优化流数据的传输和转换操作，大幅提升存储、网络和数据密集型工作负载的性能。
- **英特尔® 存内分析加速器（Intel® In-Memory Analytics Accelerator，英特尔® IAA）**可提高数据分析性能，同时从 CPU 内核上卸载任务，为数据库查询及其他工作负载加速。
- **英特尔® 动态负载均衡器（Intel® Dynamic Load Balancer，英特尔® DLB）**可随系统负载的变化将网络数据动态地分配到多个 CPU 内核上，基于硬件高效实现负载平衡。
- **面向 vRAN 的英特尔® 高级矢量扩展（Intel® Advanced Vector Extensions，英特尔® AVX）**在相同功耗范围内可将虚拟无线接入网络（vRAN）的密度较上一代产品提高多达 2 倍[2]。
- **英特尔® 高级矢量扩展 512（Intel® Advanced Vector Extensions 512，英特尔® AVX-512）**支持多达两个融合乘加（FMA）单元，并包含多项优化，可为要求严苛的计算任务提高性能。
- **英特尔® 数据保护与压缩加速技术（英特尔® QAT）**可加速加解密和数据压缩，它通过从处理器内核卸载这些任务，帮助降低系统资源消耗。
- **英特尔® 密码操作硬件加速（Intel® Crypto Acceleration）**降低了实施普遍数据加密的影响，并提高了安全套接字层（SSL）Web 服务器、5G 基础设施和 VPN/防火墙等加密敏感型工作负载的性能。

英特尔® 至强® 铂金处理器	英特尔® 至强® 金牌处理器	英特尔® 至强® 银牌处理器
多达 8 路的可扩展性	多达 4 路的可扩展性	多达 2 路的可扩展性
4 个英特尔® UPI 端口，速率为 16 GT/s	3 个英特尔® UPI 端口，速率为 16 GT/s	2 个英特尔® UPI 端口，速率为 16 GT/s
80 条 PCIe 5.0 通道 + CXL	80 条 PCIe 5.0 通道 + CXL	80 条 PCIe 5.0 通道 + CXL
支持 DDR5，速率高达 4,800 MT/s（每通道 1 个 DIMM）或 4,400 MT/s（每通道 2 个 DIMM）	支持 DDR5，速率高达 4,800 MT/s（每通道 1 个 DIMM）或 4,400 MT/s（每通道 2 个 DIMM）	支持 DDR5，速率高达 4,800 MT/s（每通道 1 个 DIMM）或 4,400 MT/s（每通道 2 个 DIMM）
支持英特尔® 傲腾™ 持久内存 300 系列	支持英特尔® 傲腾™ 持久内存 300 系列	英特尔® AVX-512（两个 512 位 FMA）
英特尔® AVX-512（两个 512 位 FMA）	英特尔® AVX-512（两个 512 位 FMA）	英特尔® 超线程技术和英特尔® 睿频加速技术
英特尔® 超线程技术和英特尔® 睿频加速技术	英特尔® 超线程技术和英特尔® 睿频加速技术	英特尔® 深度学习加速技术和英特尔® AMX
英特尔® AMX	英特尔® 深度学习加速技术和英特尔® AMX	英特尔® SGX 最大飞地容量高达 64 GB
英特尔® SST	英特尔® SST	可通过英特尔® QAT、英特尔® DLB、英特尔® DSA 和英特尔® IAA 加速工作负载
先进的可靠性、可用性和可维护性（RAS）	先进的 RAS	
英特尔® SGX 最大飞地容量高达 128 GB（在特定型号的 SKU 上最大飞地容量高达 512 GB）	英特尔® SGX 最大飞地容量高达 128 GB	
可通过英特尔® QAT、英特尔® DLB、英特尔® DSA 和英特尔® IAA 加速工作负载	可通过英特尔® QAT、英特尔® DLB、英特尔® DSA 和英特尔® IAA 加速工作负载	

访问链接了解更多第四代英特尔® 至强® 可扩展处理器详情
https://www.intel.cn/content/www/cn/zh/products/docs/processors/xeon-accelerated/4th-gen-xeon-scalable-processors.html

扫码了解更多第四代英特尔® 至强® 可扩展处理器详情

英特尔® 至强® CPU Max 系列

过去十年，随着人工智能技术的加入，峰值算力大幅增长，但由于在向内核传输数据时效率低，因此工作负载性能未能同步提升。英特尔® 至强® CPU Max 系列的诞生，使英特尔® 至强® 平台如虎添翼，它是英特尔唯一一个基于 x86 架构并采用高带宽内存（HBM）的 CPU 系列，可释放和加速内存密集型科学计算和 AI 工作负载。

更高带宽，更优性能

英特尔® 至强® CPU Max 系列采用全新微架构，支持一系列可提升平台能力的特性，包括更多内核、先进的 I/O 与内存子系统，以及可加速重大发现的内置加速器。英特尔® 至强® CPU Max 系列具有以下特性：

- 多达 56 个 P-core（性能核）：内核由 4 个小芯片构成，采用英特尔的嵌入式多芯片 互连桥接（EMIB）技术连接，功耗为 350 W；
- 64 GB 高带宽封装内存及 PCIe 5.0 和 CXL 1.1 I/O。英特尔® 至强® CPU Max 系列每核均具备 HBM 容量，可满足大多数常见科学计算工作负载的要求；
- 与其他 CPU 相比，在使用 Numenta 的 AI 技术进行自然语言处理时，其 HBM 优势可带来高达 20 倍的性能提升 [9]。

加速科学创新

- 英特尔® 至强® CPU Max 系列能够与英特尔® 至强® 平台实现轻松整合，不但可以获得处理要求严苛的工作负载所需的性能与能效，还可得到各种出色的内置加速器（包括英特尔® AMX，英特尔® DSA 等，具体详见第 53 页详细介绍）的助力。利用面向科学计算和 AI 工作负载的关键加速器，提高 CPU 使用效率、降低功耗、实现更高的投资回报率（ROI）。另外，由于处理器插槽（Socket）配置相同，可轻松将英特尔® 至强® CPU Max 系列处理器添加到第四代英特尔® 至强® 可扩展平台，并且在大多数部署方案中都无需更改代码。

灵活应对各种科学计算和 AI 工作负载

英特尔® 至强® CPU Max 系列处理器具备出色的灵活性，可根据工作负载的特性，在不同的内存模式或配置下运行：

- **“仅 HBM” 模式**：该模式支持内存容量需求不超过 64 GB 的工作负载以及每核 1 至 2 GB 的内存扩展能力，同时无需更改代码和另购 DDR，即可启动系统；
- **“HBM Flat” 模式**：该模式可为需要大内存容量的应用提供灵活性，它通过 HBM 和 DRAM 提供一个平面内存区域（flat memory region），适用于每核内存需求大于 2 GB 的工作负载。使用该模式时可能需要更改代码；
- **“HBM 缓存” 模式**：旨在提升内存容量需求大于 64 GB 或每核内存需求大于 2 GB 的工作负载的性能。使用该模式时，无需更改代码，且 HBM 可缓存来自 DDR 的事务。

英特尔® 至强® CPU Max 系列	
内核数	32-56
HBM2e 内存	64 GB
HBM 最大传输速率	3200 MT/s
DDR5最大传输速率	4800MT/s（1 个 DPC） 4400 MTs（2 个 DPC）
加速器	AMX，4 个英特尔® DSA
AI/ML指令	INT8 和 BFLOAT16

跨多架构加速科学计算和 AI 工作负载

- 整个英特尔® 至强® CPU Max 系列的产品均得到 oneAPI 的支持。oneAPI 是一个统一的、基于标准的开放式通用编程模型，可释放生产力并解锁性能。开发人员可利用英特尔® oneAPI 工具套件以及面向特定领域的专用工具套件，打造跨多种架构运行的通用计算、科学计算和 AI 应用，并对其进行分析、优化和扩展。这些资源包括矢量化、多线程、多节点并行和内存优化方面的前沿技术，可轻松构建随时能为科学计算所用的高性能、多架构软件。

访问链接了解更多第四代英特尔® 至强® CPU Max 系列详情
https://www.intel.cn/content/www/cn/zh/products/docs/processors/xeon/xeon-max-series-product-brief.html

扫码了解更多英特尔® 至强® CPU Max 系列详情

扫码了解英特尔® 至强® CPU Max 系列配置和调优指南

[9] https://www.intel.cn/content/www/cn/zh/products/docs/processors/xeon/xeon-max-series-product-brief.html

英特尔® 高级矢量扩展 512（英特尔® AVX-512）

快速分析日益增多的数据，并将其转化为有价值的洞察力，这种能力将为商业味、科学研究乃至人们的日常生活创造新的机遇。英特尔® 至强® 可扩展处理器和英特尔® 至强融核™ 处理器产品家族，增添了旨在加速数据分析的创新功能。

当前的工作负载，通常需要在多个数据元素上执行同样的操作，在传统的“标量处理”时代，指令在同一时间，只能在一个单一数据元素上执行，以致在处理海量数据时极为耗时。认识到标量处理的不足之后，从上世纪 90 年代后期开始，英特尔开始将单指令多数据流（Single Instruction, Multiple Data，SIMD）矢量功能整合到英特尔® 处理器中。英特尔® SSE 技术刚推出时，提供了 128 位寄存器和 SIMD 指令，可同时处理多达 4 个 32 位数据元素，大大加快了相关操作的处理速度。在此之后，英特尔® AVX 指令集和英特尔® AVX 2 指令集又将寄存器宽度扩展了一倍，使相关操作的处理性能实现近乎翻倍的提升。

如今，英特尔® AVX – 512 指令集将矢量计算性能提升至新高度，寄存器的宽度和数量又在英特尔® AVX 指令集和英特尔® AVX 2 指令集的基础上扩展了一倍，寄存器已由最初的 64 位升级到了 512 位，且具备两个 512 位的 FMA 单元，这意味着应用程序可同时执行 32 次双精度、64 次单精度浮点运算，或操作八个 64 位和十六个 32 位整数。

英特尔® 至强® 可扩展处理器可支持多种工作负载。英特尔® AVX-512 指令集通过矢量化性能提升，使更大数据集上的运算速度更快，满足包括科学计算在内的严苛计算任务的性能提升。例如在 OpenFOAM 在运行时，每个内核同时使用两个矢量处理单元（其中每个单元能同时处理 16 个单精度（32 位）或 8 个双精度（64 位）的浮点数）。

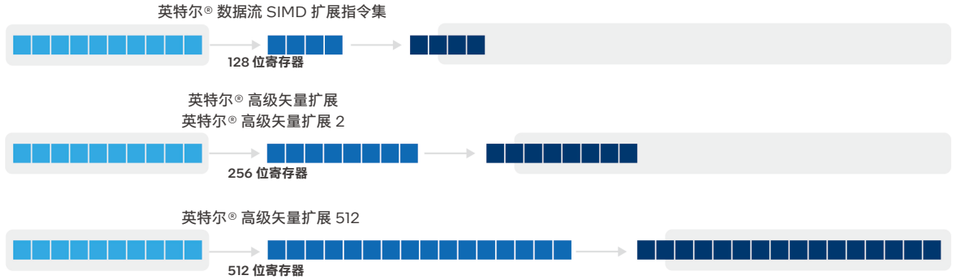


图 4-1 英特尔® SSE、英特尔 AVX2 和英特尔® AVX-512 之间的寄存器大小和计算效率的差异说明

英特尔® oneAPI 工具套件

英特尔® oneAPI 工具套件是基于新一代标准的英特尔软件开发工具，用于跨各种架构构建和部署以数据为中心的高性能的应用程序。它能够通过充分利用一流的硬件特性加速计算进程，并全面兼容现有的编程模型和代码库，可确保开发者已经编写的应用能够在 oneAPI 上无缝运行。此外，开发者只需一个代码库，便可以将应用轻松迁移到新系统和加速器上，大幅缩短了迁移时间，减轻了迁移工作量。

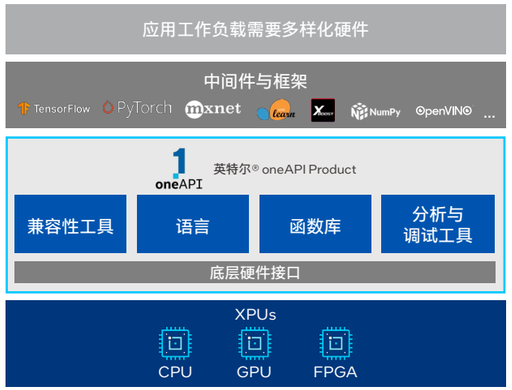


图 4-2 英特尔® oneAPI 工具套件架构

通过英特尔® oneAPI 工具套件，开发者能够使用一种通用、开放且基于行业标准的编程模型访问英特尔® CPU/GPU/FPGA。这不仅能够释放底层硬件的性能潜力，同时能降低软件开发和维护成本，并且在部署加速计算方面，英特尔® oneAPI 工具套件与专用的、受限于特定厂商的方案相比风险更低。

英特尔® oneAPI 工具套件充分利用了先进的硬件性能和指令，如用于 CPU 的英特尔® AVX-512 和英特尔® DL Boost，以及 XPU 独有的功能。英特尔® oneAPI 工具套件基于经受过长久考验的英特尔开发者工具，为开发者提供熟悉的编程语言和标准，同时与现有代码保持完全的连续性，其包括英特尔® oneAPI Base 工具包、英特尔® oneAPI AI Analytics 工具包、英特尔® oneAPI HPC 工具包及 OpenVINO™ 工具套件等不同工具。

英特尔® oneAPI HPC 工具包

- HPC 是人工智能、机器学习和深度学习应用的核心。英特尔® oneAPI HPC 工具包利用矢量化、多线程、多节点并行化和内存优化方面的最新技术，为开发人员提供了构建、分析、优化和扩展科学计算应用所需的工具。
- 该工具包是英特尔® oneAPI Base 工具包的附加组件，需要使用英特尔® oneAPI Base 工具包才能实现全部功能。它包括功能强大的以数据为中心的库、高级分析工具以及面英特尔® Python 发行版，用于为核心的 Python 数值，科学和机器学习软件包提供准原生代码级性能。

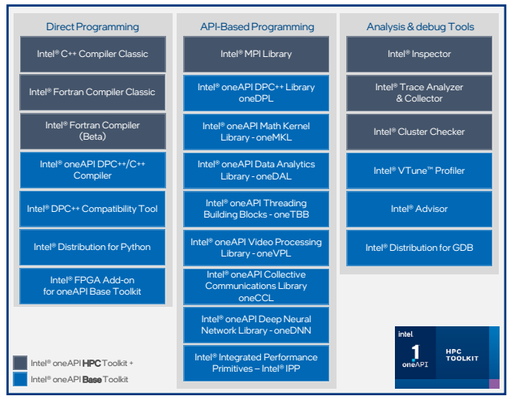


图 4-3 英特尔® oneAPI HPC 工具包

访问链接了解更多
英特尔® oneAPI 工具套件详情
https://www.intel.com/content/www/us/en/developer/tools/oneapi/toolkits.html

扫码了解更多英特尔® oneAPI 工具套件详情

英特尔® MPI 库

英特尔® MPI 库是一个多结构消息传递库，实现了开源 MPICH 规范。使用该库可以创建、维护和测试高级、复杂的应用程序，使其在基于英特尔® 处理器和兼容处理器的科学计算集群上运行得更好。即：

- 开发可在互连的多集群上运行的应用程序，用户可在启动时予以选择。
- 无需更改软件或运行环境，即可快速为最终用户提供更高性能。
- 通过自动调整实现更佳时延、带宽和可扩展性。
- 通过库链接并部署到最新的优化结构上，缩短产品上市时间。

英特尔® MPI 库特性：

支持 OpenFabrics 接口（OFI）：该优化框架为 HPC 应用提供通信服务。主要组件包括众多 API、provider 库、Kernel 服务、守护程序和测试应用。

独立互联：该库为通过 OFI 进行快速互联提供了一个经加速的，通用的多结构层，包括这些配置：

- 传输控制协议（TCP）套接字
- 共享内存
- 基于远程直接内存访问（RDMA）的互连，包括以太网和 InfiniBand

为此，它只在需要时动态建立连接，从而减少了内存占用。它还会自动选择可用的最快传输方式。

- 开发独立于结构的 MPI 代码，知道它将在运行时选择的任何网络上高效运行。
- 使用两阶段通信缓冲区放大功能，只分配所需的内存空间。

可扩展性：该库在多个结构上实现了高性能 MPI 3.1 标准，使得用户无需对软件或操作系统进行重大修改，就能快速提供更高的应用性能（即使用户更换或升级到新的互连）。

- 线程安全允许用户跟踪混合多线程 MPI 应用程序，从而在多核和众核英特尔® 架构上实现更佳性能。
- mpiexec.hydra 进程管理器提高了启动可扩展性，包括：
- 内置云服务支持，包括 Amazon Web Services, Microsoft Azure, 以及 Google 云平台

应用程序二进制接口兼容性：应用程序二进制接口（ABI）是两个程序模块之间的底层纽带。它决定了函数的调用方式，以及数据类型的大小、布局和对齐方式。有了 ABI 兼容性，应用程序就能遵守同一套运行时命名约定。

英特尔® MPI 库提供与现有 MPI-1.x 和 MPI-2.x 应用程序的 ABI 兼容性。因此，即使用户还没有准备好迁移到新的 3.1 标准，也可以通过使用该库的运行时来提高性能，而无需重新编译。

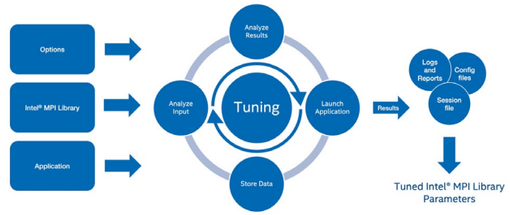
性能和调优实用程序

```
$ mpirun -n 2 IMB-MPI1 pingpong
#------------------------------------------------------------
#    Intel(R) MPI Benchmarks 2019 Update 7, MPI-1 part
#------------------------------------------------------------
# Date                  : Tue Dec  1 06:58:50 2020
# Machine               : x86_64
# System                : Linux
# Release               : 4.18.0-240.1.1.el8_3.crt1.x86_64
# Version               : #1 SMP Tue Nov 17 10:49:23 MST 2020
# MPI Version           : 3.1
# MPI Thread Environment:

# Calling sequence was:

# IMB-MPI1 pingpong
```

英特尔® MPI 基准测试是一套 MPI 性能测量工具，用于测量各种报文大小的点对点和全局通信操作。运行所有支持的基准测试，或在命令行中指定单个可执行文件，可获得特定子集的结果。

通过基准测试结果，使用者可获知：

- 集群及节点的性能
- 网络延迟和吞吐量
- MPI 执行效率



图：MPI 调优流程示意图

该库有一套强大的默认参数，用户可以按原样使用，也可以对其进行改进以确保更高性能。如果要调整默认参数之外的参数，可使用 mpitune 调整集群或应用程序参数，然后反复调整和微调参数，直到达到更加性能。

访问链接了解更多英特尔® MPI 库详情
https://www.intel.com/content/www/us/en/developer/tools/oneapi/mpi-library.html

扫码了解更多英特尔® MPI 库详情

英特尔® oneAPI 数学内核库（oneMKL）

oneMKL 是高度优化、快速、完整的数学函数库，常用于科学、工程和金融应用。oneMKL 定义了一套用于科学计算和其他应用的基本数学程序。作为 oneAPI 的一部分，oneMKL 允许在包括 CPU、GPU、FPGA 和其他加速器等各种计算设备上运行。

oneMKL 能够加快数学处理程序，提高应用性能，并减少开发时间，其具备如下特点：

- 增强的数学程序使开发人员和数据科学家能够创建高性能的科学、工程或金融应用程序；
- 核心功能包括 BLAS、LAPACK、稀疏求解器、快速傅里叶变换（FFT）、随机数生成器功能（RNG）、汇总统计、数据拟合和矢量数学；
- 针对下一代 CPU 和 GPU 进行了额外矩阵乘法优化；同时，增加了 CUDA 库函数 API 对 BLAS、LAPACK、稀疏 BLAS、向量数学、汇总统计、样条等的兼容覆盖，简化了代码向 oneAPI 和英特尔® GPU 的迁移。
- 支持第四代英特尔® 至强® 可扩展处理器的英特尔® AMX bfloat16 数据类型和英英特尔® AVX-512 bfloat16 数据类型。
- 对于以前的英特尔® 数学内核库（Intel® MKL）用户来说，是一种无缝升级。

访问链接了解更多英特尔® oneAPI 数学内核库（oneMKL）详情
https://www.intel.cn/content/www/cn/zh/developer/tools/oneapi/onemkl.html

扫码了解更多英特尔® oneAPI 数学内核库（oneMKL）详情

基于 LLVM 的英特尔® 编译器

作为一个开源的编译器框架，LLVM（Low Level Virtual Machine，底层虚拟机）已成为许多行业标准商业编译器的基础，其具有编译时间更短、移植性和灵活性更强、更便于优化和维护等优势。

英特尔对 LLVM 也有着良好的支持，在英特尔® oneAPI 中，已基于 LLVM 提供了英特尔® oneAPI DPC++/C++ 编译器和英特尔® Fortran 编译器（ifx）两个产品。其中：

- 英特尔® oneAPI DPC++/C++ 编译器可用于为 CPU、GPU、FPGA 和其他加速硬件编译数据并行 C++ 代码。编译器同时支持 Windows 和 Linux 操作系统。其构建在 LLVM 上，并使用 Clang 前端、SYCL 2020 标准通过同一源文件支持 C++ 和 OpenCL 内核。
- 英特尔® Fortran 编译器（ifx）是一个使用 LLVM 后端编译器技术的 Fortran 编译器，但其基于英特尔® Fortran 编译器经典版（ifort）前端和运行库。其有一个基本模式，支持 Fortran 77、高达 Fortran 95 的语言标准，以及 Fortran 2003 到 Fortran 2018 中的大多数功能。它还支持 OpenMP 5.0/5.1 和 OpenMP 4.5 卸载功能和指令。

更多英特尔® oneAPI DPC++/C++ 编译器和英特尔® Fortran 编译器（ifx），请参阅接下来的章节。

访问链接了解更多基于 LVMM 的英特尔® 编译器详情
https://www.intel.com/content/www/us/en/developer/articles/technical/getting-to-know-llvm-based-oneapi-compilers.html

扫码了解更多基于 LVMM 的英特尔® 编译器详情

英特尔® oneAPI DPC++/C++ 编译器

英特尔® oneAPI DPC++/C++ 编译器能够为面向未来的编程模式提供自由选择。

使用 DPC++/C++ 编译器，用户可以实现：

- 编译 ISO C++ 和 SYCL（来自 Khronos 集团）
- 跨硬件平台复用代码，包括 CPU、GPU 和 FPGA。
- 采用跨行业、开放式、基于标准的统一编程模型，避免专利锁定。

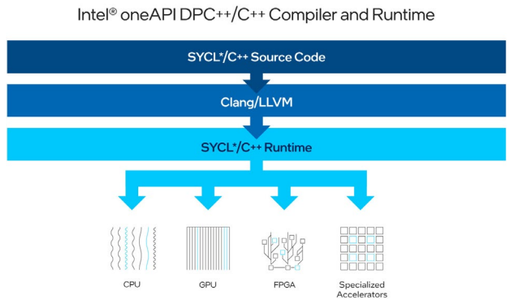
实现所有硬件价值

- 从行业领先的英特尔® 编译器技术中获得卓越性能。
- 生成优化的二进制主机代码和加速器代码。
- 使用经优化的英特尔® oneAPI 性能和线程库。

快速、正确地开发高性能代码

- 与流行的第三方编译器、开发环境和操作系统无缝集成。
- 使用最新的标准，包括用于 GPU 卸载的 C++ 20、SYCL 和 OpenMP* 5.0 和 5.1。
- 使用英特尔® C++ 编译器 classic 版，与现有的以 CPU 为核心的工作流程保持兼容

- **支持行业标准：** 支持用于 GPU 卸载的 OpenMP 4.5、5.0 和 5.1 子集。
- **可信技术：** 使用久经考验，与英特尔在编译器领域同具领先地位的 LLVM 编译器技术。



英特尔® oneAPI DPC++/C++ 编译器特性

跨架构编译

- **开发效率：** 基于单一源代码对 CPU 和加速器进行编译，同时允许自定义调优
- **语言标准支持：** 按照最新标准构建，包括 C++ 17 和 SYCL（仅适用于英特尔 oneAPI DPC++/C++ 编译器），并支持 C++ 20，以确保可迁移性，同时支持通用 lambda 表达式和变量模板等功能；

访问链接了解更多英特尔® oneAPI DPC++ 编译器详情
https://www.intel.cn/content/www/cn/zh/developer/tools/oneapi/dpc-compiler.html

扫码了解更多英特尔® oneAPI DPC++ 编译器详情

英特尔® Fortran 编译器

英特尔® Fortran 编译器在生成支持行业标准的优化代码方面有着悠久的历史，充分利用了英特尔® 至强® 可扩展处理器和英特尔® 酷睿™ 处理器的内置技术。该编译器有两个版本，分别为能够提供 CPU 和 GPU 卸载支持的英特尔® Fortran 编译器（ifx）和与现有的以 CPU 为核心的工作流程保持一致的英特尔® Fortran 编译器经典版（ifort），两个版本均可与流行的第三方编译器、开发环境和操作系统无缝集成。

为了与英特尔不断发展的多样化架构保持一致，该编译器现在还支持图形处理器。使用这个支持 OpenMP 且基于标准的 Fortran 编译器可实现 CPU 和 GPU 分载。

英特尔® Fortran 编译器（ifx）基于英特尔® Fortran 编译器经典版（ifort）前端和运行库，但使用了 LLVM 后端编译器技术。ifx 兼容二进制文件（.o/.obj）和模块文件（.mod）。用 ifort 生成的二进制文件和库可与用 ifx 生成的二进制文件和库链接，用一种编译器生成的 .mod 文件可用于另一种编译器（仅限 64 位目标）。两种编译器使用相同的运行时库。ifx 的性能可能与 ifort 编译的应用程序相当，也可能不相当。在整个 2023 年，ifx 的性能将随着每次更新版本的发布而提高。

访问链接了解更多
英特尔® Fortran 编译器详情
https://www.intel.cn/content/www/cn/zh/developer/articles/release-notes/oneapi-fortran-compiler-release-notes.html

扫码了解更多英特尔® Fortran 编译器详情

英特尔® Trace Analyzer and Collector（ITAC）

英特尔® 跟踪分析器和收集器（Intel® Trace Analyzer and Collector，ITAC）是一款出色的工具，可帮助客户调试 MPI、消息传递接口应用程序：

- 查找代码中的时间依赖性和瓶颈；
- 检查应用程序的正确性；
- 定位潜在的编程错误、缓冲区重叠以及死锁
- 可视化解析并行应用程序行为；
- 评估统计和负载平衡；
- 分析子程序或代码块的性能；
- 了解通信模式、参数和性能数据；
- 识别通信热点；
- 缩短解决问题的时间，提高程序应用效率。

访问链接了解更多
英特尔® 跟踪分析器和收集器详情
https://www.intel.com/content/www/us/en/developer/tools/oneapi/trace-analyzer.html

扫码了解更多英特尔® 跟踪分析器和收集器详情

英特尔® vTune™ Profiler

intel VTune Profiler

英特尔® VTune™ Profiler 可面向科学计算、云、物联网、媒体、存储等工作负载，针对应用性能、系统性能和系统配置等进行优化。

- CPU、GPU 和 FPGA：针对整个应用的性能进行调优，而不仅仅是加速部分。
- 多语言：支持 SYCL、C、C++、C#、Fortran、OpenCL™ 代码、Python、Google Go 编程语言、Java、.NET、Assembly 或任何语言组合的配置文件。
- 系统或应用程序： 持续获取粗粒度系统数据或映射到源代码的详细结果。
- 功率： 优化性能，同时避免功率和热相关节流。

英特尔® VTune™ Profiler 特性

算法优化

- 查找热点 -- 代码中最耗时的部分。
- 使用 Flame Graph 可视化查看热点代码路径和每个函数及其调用所花费的时间。

微架构和内存瓶颈

- 通过微架构探索分析，找出影响应用程序性能的最关键的硬件问题。
- 针对内存访问相关的问题，如缓存未命中和高带宽问题。

加速器和 XPU

- 针对 SYCL、OpenCL 代码、Microsoft DirectX 或 OpenMP，为 GPU 卸载模式和数据传输提供优化。
- 分析 GPU 绑定代码，找出因微架构限制或低效内核算法造成的性能瓶颈。
- 探索 CPU 与 FPGA 的交互以及 FPGA 的使用。

并行性

- 检查代码线程化的效率，确定影响性能的线程问题。
- 评估计算密集型或吞吐量大的科学计算应用程序，以确保高效使用 CPU、矢量化和内存。

平台和 I/O

- 找出 I/O 密集型应用的性能瓶颈。了解硬件如何有效处理外部 PCIe 设备或集成加速器产生的 I/O 流量。
- 利用 Platform Profiler 查看长期运行工作负载的系统运行状态。
- 利用 “系统概述” 对短期运行的工作负载进行细粒度概述。

多节点

- 描述 MPI 和 OpenMP 工作负载的性能。
- 明确可扩展性问题，并提供深入分析和建议。

访问链接了解更多
英特尔® VTune™ Profiler 编译器详情
https://www.intel.cn/content/www/cn/zh/developer/tools/oneapi/vtune-profiler.html

扫码了解更多英特尔® VTune™ Profiler 编译器详情

英文全称	英文缩写	中文全称
Algebraic multigrid	AMG	代数多重网格
Application Binary Interface	ABI	应用程序二进制接口
Artificial Intelligence	AI	人工智能
Basic Linear Algebra Subprograms	BLAS	基础线性代数子程序库
Burrows-Wheeler Aligner	BWA	
Cluster File System	CFS	集群文件系统
Compound Annual Growth Rate	CAGR	年复合增长率
Computational Fluid Dynamics	CFD	计算流体动力学
Compute Express Link	CXL	
Computer Aided Engineering	CAE	计算机辅助工程
Cryo-electron Microscopy	Cryo-EM	冷冻电子显微镜
Density Functional Theory	DFT	密度泛函理论
Double-precision		双精度
Dual Inline Memory Module	DIMM	双列直插内存模块
Fast Fourier transform	FFT	快速傅里叶变换
Fastest Fourier Transform in the west	FFTw	
Finite Element Analysis	FEA	有限元分析
Flat Memory Region		平面内存区域
Gaussian Elimination		高斯消元法
Gauss-Seidel		高斯 - 塞德尔
Genomic Analysis Toolkit	GATK	基因组分析工具套件
High Bandwidth Memory	HBM	高带宽内存
High Performance Conjugate Gradient	HPCG	高性能共轭梯度
High Performance Linpack	HPL	高性能 Linpack
High Rank Count		高秩计数
High-throughput Sequencing	HTS	高通量测序
Instruction Set Architecture	ISA	指令集架构
Intel® C++ Compiler	ICC	英特尔® C++ 编译器
Intel® Advanced Matrix Extensions	Intel® AMX	英特尔® 高级矩阵扩展
Intel® Advanced Vector Extensions	Intel® AVX	英特尔® 高级矢量扩展
Intel® Advanced Vector Extensions 512	Intel® AVX - 512	英特尔® 高级矢量扩展 512
Intel® Branch		英特尔® 分支
Intel® Crypto Acceleration		英特尔® 密码操作硬件加速
Intel® Data Streaming Accelerator	Intel® DSA	英特尔® 数据流加速器
Intel® Dynamic Load Balancer	Intel® DLB	英特尔® 动态负载均衡器
Intel® Embedded Multi-Die Interconnect Bridge	Intel® EMIB	英特尔® 嵌入式多芯片互连桥接
Intel® Fork		英特尔® 分叉
Intel® Hyper-Threading Technology		英特尔® 超线程技术
Intel® Infrastructure Processing Unit	Intel® IPU	英特尔® IPU
Intel® In-Memory Analytics Accelerator	Intel® IAA	英特尔® 存内分析加速器

英文全称	英文缩写	中文全称
Intel® QuickAssist Technology	Intel® QAT	英特尔® 数据保护与压缩加速技术
Intel® Speed Select Technology	Intel® SST	英特尔® SST
Intel® System Configuration Utility		英特尔® 系统配置实用程序
Intel® Threading Building Blocks	Intel® TBB	英特尔® 线程构建模块库
Intel® Turbo Boost Technology		英特尔® 睿频加速技术
Intel® Ultra Path Interconnect	Intel® UPI	英特尔® 超级通道互联
Intel® Virtual RAID on CPU	Intel® VROC	英特尔® VROC
Java Runtime Environment	JRE	Java 运行环境
Large-scale Atomic/Molecular Massively Parallel Simulator	LAMMPS	大规模原子分子并行模拟器
Linear System Package	Linpack	线性系统软件包
Long-Range Thread	LRT	长线程
Low Level Virtual Machine	LLVM	底层虚拟机
Low Rank Count		低秩计数
Molecular Dynamics	MD	分子动力学
Molecular Systems with Long-range Electrostatics		长距离静电分子系统
NAnoscale Molecular Dynamics	NAMD	
Non Uniform Memory Access	NUMA	非一致性内存访问
Open source Field Operation And Manipulation	OpenFOAM	
Particle – Particle-Particle – Mesh	PPPM	质点 - 质点 - 质点 - 网格
Preconditioned Conjugate Gradient	PCG	预处理共轭梯度
Projector Augmented Wave	PAW	投影缀加波
Random Number Generator	RNG	随机数生成器
Remote Direct Memory Access	RDMA	远程直接内存访问
Return on Investment	ROI	投资回报率
Single Instruction Multiple Data	SIMD	单指令多数据
Single-precision		单精度
Smoother		平滑器
Socket		处理器插槽
Solid State Disk	SSD	固态盘
Sparse Matrix-Vector Multiplication	SpMV	稀疏矩阵向量乘法
Sparse triangular solve	TRSV	稀疏三角形解
Sub-Numa Cluster	SNC	Sub-Numa 集群
Symmetric Gauss-Seidel smoother	SYMGS	对称高斯 - 塞德尔平滑器
Tool Command Language	TCL	工具命令语言
Total Rank Count		总秩计数
Transmission Control Protocol	TCP	传输控制协议
Transparent Huge Pages	THP	透明大页
Ultrasoft Vanderbilt Pseudopotentials	US-PP	超软赝势
Vienna Ab-Initio Simulation Package	VASP	

扫码访问英特尔官网
了解更多英特尔在科学计算领域的技术实践

扫码下载
《英特尔中国科学计算实战手册》

免责声明：

性能测试中使用的软件和工作负荷可能仅在英特尔微处理器上进行了性能优化。诸如 SYSmark 和 MobileMark 等测试均系基于特定计算机系统、硬件、软件、操作系统及功能。上述任何要素的变动都有可能导致测试结果的变化。请参考其他信息及性能测试（包括结合其他产品使用时的运行性能）以对目标产品进行全面评估。更多信息，详见 www.intel.com/benchmarks。

在特定系统的特殊测试中测试组件性能。硬件、软件或配置的差异将影响实际性能。当您考虑采购时，请查阅其他信息来源评估性能。关于性能和基准测试程序结果的更多信息，请访问 www.intel.com/benchmarks。

英特尔技术特性和优势取决于系统配置，并可能需要支持的硬件、软件或服务得以激活。产品性能会基于系统配置有所变化。没有任何产品或组件是绝对安全的。更多信息请从原始设备制造商或零售商处获得，或请见 intel.com。

优化声明：英特尔编译器针对英特尔微处理器的优化程度可能与针对非英特尔微处理器的优化程度不同。这些优化包括 SSE2、SSE3 和 SSSE3 指令集和其他优化。对于非英特尔微处理器上的任何优化是否存在、其功能或效力，英特尔不做任何保证。本产品中取决于微处理器的优化是针对英特尔微处理器。不具体针对英特尔微架构的特定优化为英特尔微处理器保留。请参考适用的产品用户与参考指南，获取有关本声明中具体指令集的更多信息。

没有任何产品或组件是绝对安全的。

描述的成本降低情景均旨在在特定情况和配置中举例说明特定英特尔产品如何影响未来成本并提供成本节约。情况均不同。英特尔不保证任何成本或成本降低。

英特尔并不控制或审计第三方数据。请您审查该内容，咨询其他来源，并确认提及数据是否准确。