网格应用开发环境设计需求文档

(IDE for Developing Mesh-based Applications)

1. 简述

基本目标是设计一个能够开发 Mesh 应用的 IDE。适合于巨大网格结构; 在 PC、一般集群和超级计算机均可运行;并且在 PC 机上开发的程序和数据等可以在超大机器上直接运行。

2. 体系结构配置生成与编辑

2.1 功能简述

① 获得当前机器的配置
② 编辑一个指定的配置(文件)
③ 保存机器配置到 HDF5 文件(定义保存格式)
④ 自动获取当前机器的配置信息并且输出固定格式的配置文件。
⑤

2.2 详细功能需求

① 定义体系结构配置文件的逻辑格式和存储结构
② 存储结构用 HDF5 实现
③ 能打开,编辑、保存配置文件
④ 配置文件的内容暂包括下列所能获得的内容
⑤ 获得当前系统的节点数及其逻辑拓扑连接及其连接的平均带宽
⑥ 配置
   a) 配置文件和网格文件有关联关系，由不同配置文件配置都有可能，网格应用程序配置之间的关系（网格的存储依赖配置，会随着变化，一对一的，粗化和配置的关系，反映实际的计算更强烈一些）
   b) 很大的网格要一点一点读取
   c) 配置可以改变换机器的，粗化可以是基于分布的，可以拷贝过去到天河
   d) refinement 良化（但是保持精度），但是粗化（效率），计算密度高的地方要注意会不会被划分

⑦ 获得每个节点的的配置
   a) 获得 CPU 核数，频率，SIMD 宽度，context 数，SMT 数，存储器带宽
   b) 各级 Cache 尺寸大小，line 大小，是否有一致性机制 ac（可以省略）
   c) 吞吐量
   d) 显存容量
⑧ 获得操作系统及相关库的信息
   a) 软件安装位置

b) 标准库位置
c) 需要安装组件目录
d) 打开区域支持. 区域支持会带来性能损失，如果不是在说英语的环境使用数据库，那么你很可能需要这个选项
e) 制作 C 接口模块，安装到通常的安装位置（典型的位置是/usr/lib/C），所以必须要有 root 权限来执行安装步骤）制作那些需要 Tcl/Tk 的部件，就是 libpgtcl，pgtclsh，pgtksh，pgaccess 和 PL/Tcl.
f) 制作 ODBC 驱动包（开放式数据库连接性）

⑨ 参考链接：linux ./configure 的参数详解

2.3 非功能需求

获得 CPU 温度
显示器分辨率
驱动版本

3. 网格结构的转换与存储

3.1 功能简述

① 将一个文本格式的网格结构按指定的一个配置(或省缺配置)转换(划分)为 HDF5 格式（HDF5 是分布格式,需定义或依据 OP2 的定义）
② 将一个(文本格式或 HDF5 格式)网格结构粗化
③ 将一个配置的分布网格转换为另一个配置的分布网格结构(算法)

3.2 详细功能需求

1. 存储整个网格的结构及数据。
2. 存储划分后的网格结构及数据（分布式的）。
3. 网格结构及数据可存储在文本文件或 HDF5 文件（分布式）中。
4. 可从文本文件和 HDF5 文件加载网格。
5. 如果网格是分布存储，则并行访问 HDF5 文件，每个计算节点访问不同的部分。
6. 需定义分布存储的格式（包含体系结构的配置,省缺为均衡的配置）。
7. 可从 HDF5 文件中读取配置信息。
8. 文本格式的网格可以转换为 HDF5 格式的网格。
9. 网格文件格式的转换可按一个指定的配置（或省缺配置）进行。
10. 将文本格式或 HDF5 格式的网格结构粗化，使得粗化网格程序能够在普通 PC 机上快速运行，便于用户快速调试程序。
11. 可将未划分的整个网格根据配置划分为分布式网格。
12. 可将一个配置的分布网格转换为另一个配置的分布网格结构。

3.3 非功能需求

1. 粗化网格应尽量保持原网格的结构及数据特性。
2. 网格结构的转换、粗化、划分过程耗时应尽量短。

4. 网格应用开发环境及其工具链(IDE)

4.1 功能简述

① 编辑一个网格应用程序
② 将一个网格应用编译到目标并行程序并建立对应项目管理文件(见 5)
③ 优化目标应用程序(见 6)
④ 调试运行目标应用程序(使用已有的调试器)
⑤ 构建目标并行应用程序(见 7)
⑥ parallel studio 工具链

4.2 详细功能需求

1. 可新建文件（项目），也可打开已有文件（包括在线文件）。
2. 编辑窗口可直接编辑源代码，方向键 tab 等快捷键默认设置同 codeblocks，也允许用户自行设置习惯用法。
3. 自动根据语言类型库补全，以及项目模板帮助初学者使用。
4. 支持并行构建(利用 CPU 的额外内核)。
5. 可高亮显示，同时显示行号，块区分标记等。
6. 可更改界面风格。
7. 搜索和查找，替换，标记关键字等。
8. 编码格式——UTF－8，ANSI，等。
9. 支持插件。
11. 非常快速的自定义构建系统(可以 makefile)。
12. 用于组合多个项目的工作区。
13. 导入 MSVC 项目和工作区。
14. 完整的断点支持：代码断点，数据断点（读、写和读/写），断点条件（仅当表达式为真时才中断），断点忽略计数（仅在命中一定数量后才中断）。
15. 显示本地函数符号和参数。
16. 预编译。
17. 编译。
18. 运行。
19. 生成可执行文件。
20. 优化源代码。
21. 生成并行程序。
22. debug。
23. 可扩展安装其他来源插件。
24. 允许多线程并行操作，能够编写单个程序内置于三个不同的可执行文件中，用于不同的单节点。
25. 有些用户不喜欢使用 HDF5 文件，或者至少不在 op2 规定的方式。为了支持这些用户，应用程序代码可以执行自己的文件 I/O，然后使用标准程序。

4.3 非功能需求

1. 性能因素考虑。
2. 支持多种编程语言，提供编写编译调试运行的功能，并且简化步骤，简化编写和调试任务，尽可能提供优良的运行功能。
3. 专业简单直观的界面和分类。
4. 计算结果精准。
5. 允许吞吐大量数据。
6. 支持多个系统，可以在 x86 系统和 Linux 上运行。

5. 并行化及优化编译器(Parallelizing Compiler)

5.1 功能简述

① 将用户的网格应用源码编译成并行的应用程序源代码，(目标并行程序的平台和程序模型?)
② 编译过程支持初步优化(面向体系结构的负载均衡,基本的多线程化,基本的向量化)
③ 并行的目标源代码语言及其并行编程模型尽量统一
④ 支持异构的目标系统

5.2 详细功能需求

① 转换过程要进行体系结构相关的负载均衡
② 要对每一个不同的体系结构计算结点做第二次并行化和向量化
③ 尽量采用统一的并行编程模型(通信子,线程组,任务集)
④ 尽量采用统一的目标编程语言(MPI,openMP4,SIMD 三层)
⑤ 平台用 SIMD， OPENMP(4.5 以上，支持 GPU)， MPI 组合或者直接用 MPI 加 openMP
⑥ 纯 MPI 最新版(需要测试)
⑦ SIMD 向量化
⑧ 使用 openMP
⑨ GPU 加速器考虑
⑩ 看纯粹 mpi openmp 是否支持，occ ocpl
⑪ cuda 编译目标
⑫ 异构系统
⑬ CPU 逻辑计算， 加速器系统之间有差异
⑭ 多进程多线程向量化
⑮ 暂时不考虑优化
⑯ 在 Pc 配置上映射做负载均衡，计算节点复杂度和交互的带宽，效率模型，性能模型，代码形式
⑰ 先进的优化,单指令多数据(SIMD)向量化,和循环记忆转换
⑱ 集成性能库
⑲ 使用最新的 OpenMP *并行编程模型
⑳ 内存管理的改进,减少应用程序编译时不牺牲运行时性能
㉑ 完整的 OpenMP 4.5 支持和初始 OpenMP 5.0 支持
㉒ 支持一个用户定义的感应 OpenMP 平行的语法和 SIMD 独家扫描
㉓ c++完整 c++ 11 日 14 日和初始 c++ 17 的支持 （修改为类 C 语言支持）
㉔ SIMD 数据布局模板(SDLT)标准库
㉕ array-of-structure 代码进行向量化

㉖ 虚函数向量化来提高面向对象的（c++）代码的性能
㉗ 改进λ和常数表达式支持（parallel studio 特点补充）
㉘ 增强兼容新版本的GNU c++和Microsoft Visual c++编译器（类C的op2编译工具）
㉙ 提升单指令多数据(SIMD)向量化和线程功能
㉚ 使用最新的OpenMP *并行编程模型
㉛ 利用更多的内核和增加向量寄存器的宽度
㉜ 符合开放标准,如OpenMP
㉝ 使用熟悉的工具在首选平台: Linux

分几种情况
按照是否人工干预分：

全自动并行化：编译器自动或半自动地将串行程序并行化
采用过程间分析、符号数据相关性分析、数组私有化、归约识 别、复杂形式的归纳变量识别以及运行时分析等技术，不需要人为干预，自动发现程序中可并行的部分来生成并行代码

交互式并行化：程序员显式使用并行编程技术来开发应用程序
① 给用户提供程序中的有效信息，包括相关性分析结果、程序调用图、性能预测结果等帮助用户进行并行化工作。
② 在尽可能采取自动并行化技术的同时，允许在并行化过程中人为查看和修改并行化结果，通过利用人工的能力来提高并行化效果，在部分程度上弥补 了全自动并行化系统的不足。
③ 交互式并行化插件和交互式并行化引擎，交互式并行化插件为基于编译器的交互式视图插件，通过和用户交互来获取交互信息，将串行程序信息和交互信息传输到交互式并行化引擎；
④ 交互式并行化引擎接收串行程序和交互信息，确定串行程序的计算结构特性，对不同特性的计算采用不同并行化策略进行并行化，并将并行化结果传递给交互式 并行化插件。
⑤ 在预备阶段交互式并行化插件与用户交互获取交互信息，交互式并行化引擎接收交互信息并进行自动化程序分析， 确定串行程序的计算结构特性；
⑥ 在并行化阶段，交互式并行化插件与用户交互获取交互信息，交互式并行化引擎对不同特性的计算采用不同的并行化策略进行并行化，并将并行化结果传递给交互式并行化插件，编译器显示并行化结果。
⑦ 基于编译器的插件来实现交互功能
⑧ 采用源到源的变换，具有通用性，可针对不同目标机器体系结构获得较好的并行化效果
⑨ 采用最新自动并行化分析技术与交互式结合，较好地抽取程序信息及用户知识，具有较强的程序理解能力；
⑩ 支持多种不同类型计算，提供相应的并行化方案
⑪ 支持多种粒度的并行性
⑫ 支持投机多线程并行化
⑬ 先进行粗粒度自动并行化代价评估，其次进行中粒度自动并行化代价评估，最后进行细粒度自动并行化代价评估，然后进行综合代价评估

根据并行模型分：

1. 共享内核并行(明确的线程，比如 Pthreads 和 Java threads)；
2. 共享内存并行(任务/数据的并行，比如 OpenMP)；
3. 分布式并行(明确的通信，比如 MPI, SHMEM, 和 Global Arrays)；
4. 分布式并行(特殊的全局访问，比如 Co-Array Fortran, UPC)。

并行策略：

1. 使用适当的编译器选项集，以在单个处理器上获得最佳串行性能。
2. 使用典型测试数据，确定程序的性能配置文件。标识最主要的循环进行优化。
3. 确定串行测试结果是准确的。使用这些结果以及性能配置文件作为基准。
4. 使用选项和指令组合编译并生成并行化的可执行文件。
5. 基于性能对并行化方案进行改进。
6. 自动和约简。约简操作可将数组中的元素约简为单个值
7. 在栈中分配局部变量
8. 显示并行化哪些循环
9. 显示并行化哪些循环，显示显式情况下的警告
10. 编译以实现 OpenMP 并行化
11. 对于显式并行化，编译器对于 while 等循环指令生成并行化代码，即使它包含子程序调用。
12. 检测主要抑制因素，这些因素可以防止对循环进行显式并行化
13. 在并行执行的循环中执行 I/O
14. 应用程序首先成为多线程程序。需要对可并行执行的任务进行标识并重新编程，使其计算分布在多个处理器或线程上。
15. 将关注焦点放在循环上。将循环的计算工作分配到多个处理器上，无需修改源程序。选择哪些循环进行并行化以及如何分配这些循环可以完全让编译器去决定，也可以由程序员使用源代码指令来显式指定，还可以采用组合方式来实现。
16. 分叉- 合并(Fork-Join)方法，用语言扩展(language extension) 支持，比如 Cilk Plus 、 OpenMP 等。一个程序可以分解成一系列小任务块，每个小任务块包括各自的一系列指令。互相不依赖的一组小任务块，即可考虑并行执行（分叉出一组并行的线程），然后通过合并（Join）同步，等所有进程结束后再往下运行
17. 源到源的转化（多文件到多个流文件）
18. 在 MPI 应用程序中，同一程序的多个副本执行为分离进程，通常位于计算集群的不同节点上。
19. 分布式，并行访问 HDF5 文件，每个计算节点访问不同得部分
20. 支持并行构建(利用 CPU 的额外内核)
21. 允许多线程并行操作，能够编写单个程序内置于三个不同的可执行文件中，用于不同的单节点

1. 将用户的类 C 程序（也可以看作是并行）转换成下列形式的并行程序
   ① OpenMP
   ② OpenMp+SIMD
   ③ Cuda

④ ACC
⑤ 串行的程序
(以上五点均支持 MPI)
22.

5.3 非功能需求

6. 并行化源码分析器和优化细调器(Tuner 暂省略)

6.1 功能简述

① 根据源代码及分析 trace 结果, 优化并行程序
② 优化的主要是 compile 之后的程序文件

6.2 详细功能需求

暂时不考虑：加速，向量化，cache，手动修改，线程树扩大，二级优化

1、 **英特尔 Parallel Advisor(辅助创建多线程)**
对已有的串行程序进行分析，辅助改造和设计多线程并行程序。主要有两点:1、寻找适合并行程序的点;2、提供添加并行程序的向导。

2、 **英特尔 Parallel Composer(创建器)**
此工具捆绑了“英特尔® C++ 编译器、英特尔® 线程构建模块”(英特尔® TBB)、“英特尔® 集成性能基元”及“英特尔® Parallel Debugger Extension”。此工具使得 TBB 能够更容易的和 visual studio 进行兼容，提高开发效率。

3、 **英特尔 Parallel Amplifier(分析器)**
英特尔® Parallel Amplifier 有三种分析类型，旨在让以不同角度深入了解程序性能。主要有热点分析、并发性分析和锁定和等待，主要运用在性能分析上

4、 **英特尔 Parallel Inspector(检查器)**
对已经完成的并行程序进行检查，1、检查内存的泄露和冲突问题;2、检查数据冲突和死锁问题。

6.3 非功能需求

7. 并行源码构建器(Builder, 项目构建器)(zlb)

7.1 功能简述

① 根据并行程序平台及模型调试和构建并行源码
② 可调试和构建纯 MPI(串行)、MPI+多线程、MPI+多线程+SIMD 的程序

③ 用户有一个开发工具（命令行）可以编辑自己的应用程序，被编译为并行程序，项目管理文件
④ 一个 Cmake 规范
⑤ 设计：makefile 语法　cmake 教程
⑥

7.2 详细功能需求

① 程序构建成目标程序　make　cmake 简单方便功能满足
② mpi 缺点纯粹考虑分布，同步问题是问题，最好只有一个，
③ mesh 结构的数据并行，只要处理好 kernel 函数，
④ 抽象层工具
⑤ trace 文件分析
⑥ intel　parralell　studio
⑦ 运行环境下也是 hdf5　优化 io 并行，
⑧ 手动将代码变成目标代码（使用 makefile,cmake 等工具）
⑨ 考虑编写一个生成器，符合要求的 makefile
⑩ Cmake 做一些规范，如果不使用工具的话就要实现一个规范
⑪ 构建过程不影响主要并行程序，可执行程序一定是并行的，hdf5 去调用 makefile

7.3 非功能需求

以下补充来源：parallel studio tools

8. 分布式工具 Distribution

① 实现更快的 Python 应用程序所有性能的盒子,极少或没有修改你的代码
② 加速 NumPy *, SciPy *, scikit-learn *集成英特尔®内核性能库如英特尔®数学库和英特尔®加速度数据分析库
③ 访问最新的向量化和多线程指令,Numba *和 Cython *,可组合并行线程构建块,等等

9. 内存和线程调试器分析器 Inspector

① 省钱:定位内存和线程错误的根源在你释放。
② 节省时间:快速调试断断续续的种族和死锁。
③ 保存数据:发现错误,如遗漏或冗余的缓存将持久内存的实现。
④ 保存工作:使用独立的接口,微软 Visual Studio 插件,或者命令行。不需要特殊的编译器或构建。

10. 向量化和线程顾问 Advisor

① Roofline 分析-优化您的应用程序进行内存和计算。
② 向量化优化-启用更多向量并行性并提高其效率。
③ 线程原型-对多个线程设计进行建模，调整和测试。
④ 建立异构算法-创建并分析数据流和依赖性计算图。

⑤ 视觉上比较多少每个部分的代码改进了每次使用车顶比较优化。
⑥ 套接字的数量调整屋顶值更精确的性能上限 MPI 应用程序。
⑦ 专注于重要的通过定制您的需求并保存检验报告列配置,供以后使用。
⑧ 分析 OpenMP *任务依赖图源相关的流图分析仪。

11. 性能剖析器 Amplifier

① 高性能计算(HPC)在天气预报中,有限元分析和生物信息学
② 为物联网嵌入式应用程序、运输和制造
③ 媒体视频转码和图像处理软件
④ 云应用程序或者 Java *服务容器
⑤ 设备驱动程序
⑥ 游戏引擎
⑦ 存储,包括存储性能开发工具包(SPDK)和数据平面软件开发工具包(DPDK)调查

12. 集群检查程序 checker

① 运行此工具,确保完整和配置您的系统并行运行一个应用程序。
② 找出问题点和改善集群获得详细的诊断信息功能和性能。
③ 在几分钟内解决问题,减少需要专门的支持能力。
④ 创建自定义测试的可扩展性。
⑤ 更快的默认测试
⑥ 预定义的深入测试设置为用户或 admin-specific 分析
⑦ 一个增强的汇总输出节点上列出简短的事实和问题
⑧ 故障诊断测试先决条件英特尔®MPI 库
⑨ 能力验证 BIOS 的一致性和管理英特尔®服务器的固件设置董事会通过系统配置实用程序(syscfg)
⑩ 支持最新的英特尔®Xeon®铂 9200 处理器
⑪ 集数据年龄阈值(- t)选择
⑫ Bug 修复和改善,CVE 更新

13. 使用规范设计

i. **配置读取工具：**
   1. 登录 Terminal，执行：cat /proc/cpuinfo，就会显示出主机的 CPU 详细参数，如内核、频率、型号等等
   2. 查看主机的内存信息，执行：cat /proc/meminfo，主要是看内存大小、交换空间、高速缓存
   3. top:负载，进程，cpu,内存，交换分区
   4. htop:互动进程查看
   5. dstat 显示了 cpu 使用情况，磁盘 io 情况，网络发包情况和换页情况，输出是彩色的，可读性较强，相对于 vmstat 和 iostat 的输入更加详细且较为直观
   6. 综合篇：Linux 性能测试工具

ii. **编辑器：gedit,vscode,vim，nano，emacs，sublimetext 等（Linux 环境下编辑器）**

iii. **构建器：cmake,makefile**

iv. **编译器：op2 拆分后的工具**

v. **调试工具：**

1. op2 拆分后的工具
2. gdb
3. 程序静态分析工具 splint
4. 程序执行性能分析工具 prof/gprof
5. linux 调试工具

vi. **分析器：**

==========

目标的 Mesh 应用项目管理尽量简单,但功能要强大(现在 OP2 用 cmake)

能同时运行在 PC 上和目标机器上,PC 上更主要是开发环境,目标机上主要是运行环境
可以使用 codeblocks 做为开发环境,工具可以做为插件
开发环境可以用多种语言实现,比如 Python
开发环境包括 Editor,compiler 等,可以支持对目标应用的源编译和最终目标码的编译
开发过程可以命令行方式,也可以在 IDE 中

Mesh 存储要基于 HDF5,其它格式可以转换到 HDF5
存储整个 MESH 及数据
存储划分后的 MESH 结构及数据（分布式的）
分布的话，并行访问 HDF5 文件，每个计算节点访问不同得部分
支持其他格式向 HDF5 转换
要定义分布存储的格式（包含体系结构的配置,省缺为均衡的配置）

7.2 详细功能需求

分几种情况
按照是否人工干预分：

全自动并行化：编译器自动或半自动地将串行程序并行化
采用过程间分析、符号数据相关性分析、数组私有化、归约识 别、复杂形式的归纳变量识别以及运行时分析等技术，不需要人为干预，自动发现程序中可并行的部分来生成并行代码

交互式并行化：程序员显式使用并行编程技术来开发应用程序

⑭ 给用户提供程序中的有效信息，包括相关性分析结果、程序调用图、性能预测结果等帮助用户进行并行化工作。

⑮ 在尽可能采取自动并行化技术的同时，允许在并行化过程中人为查看和修改并行化结果，通过利用人工的能力来提高并行化效果，在部分程度上弥补 了全自动并行化系统的不足。

⑯ 交互式并行化插件和交互式并行化引擎，交互式并行化插件为基于编译器的交互式视图插件，通过和用户交互来获取交互信息，将串行程序信息和交互信息传输到交互式并行化引擎；

⑰ 交互式并行化引擎接收串行程序和交互信息，确定串行程序的计算结构特性，对不同特性的计算采用不同并行化策略进行并行化，并将并行化结果传递给交互式 并行化插件。

⑱ 在预备阶段交互式并行化插件与用户交互获取交互信息，交互式并行化引擎接收交互信息并进行自动化程序分析， 确定串行程序的计算结构特性；

⑲ 在并行化阶段，交互式并行化插件与用户交互获取交互信息，交互式并行化引擎对不同特性的计算采用不同的并行化策略进行并行化，并将并行化结果传递给交互式并行化插件，编译器显示并行化结果。
⑳ 基于编译器的插件来实现交互功能
㉑ 采用源到源的变换，具有通用性，可针对不同目标机器体系结构获得较好的并行化效果
㉒ 采用最新自动并行化分析技术与交互式结合，较好地抽取程序信息及用户知识，具有较强的程序理解能力；
㉓ 支持多种不同类型计算，提供相应的并行化方案
㉔ 支持多种粒度的并行性
㉕ 支持投机多线程并行化
㉖ 先进行粗粒度自动并行化代价评估，其次进行中粒度自动并行化代价评估，最后进行细粒度自动并行化代价评估，然后进行综合代价评估

根据并行模型分：

1. 共享内核并行(明确的线程，比如 Pthreads 和 Java threads)；
2. 共享内存并行(任务/数据的并行，比如 OpenMP)；
3. 分布式并行(明确的通信，比如 MPI, SHMEM, 和 Global Arrays)；
4. 分布式并行(特殊的全局访问，比如 Co-Array Fortran, UPC)。

并行策略：

23. 使用适当的编译器选项集，以在单个处理器上获得最佳串行性能。
24. 使用典型测试数据，确定程序的性能配置文件。标识最主要的循环进行优化。
25. 确定串行测试结果是准确的。使用这些结果以及性能配置文件作为基准。
26. 使用选项和指令组合编译并生成并行化的可执行文件。
27. 基于性能对并行化方案进行改进。
28. 自动和约简。约简操作可将数组中的元素约简为单个值
29. 在栈中分配局部变量
30. 显示并行化哪些循环
31. 显示并行化哪些循环，显示显式情况下的警告
32. 编译以实现 OpenMP 并行化
33. 对于显式并行化,编译器为标有 **PARALLEL DO** 或 **DOALL** 指令的循环生成并行化代码，即使它包含子程序调用。
34. 检测主要抑制因素，这些因素可以防止对循环进行显式并行化
35. 在并行执行的循环中执行 I/O
36. 应用程序首先成为多线程程序。需要对可并行执行的任务进行标识并重新编程，使其计算分布在多个处理器或线程上。
37. 将关注焦点放在循环上。将循环的计算工作分配到多个处理器上，无需修改源程序。选择哪些循环进行并行化以及如何分配这些循环可以完全让编译器去决定，也可以由程序员使用源代码指令来显式指定，还可以采用组合方式来实现。
38. 分叉- 合并(Fork-Join)方法，用语言扩展(language extension) 支持，比如 Cilk Plus 、 OpenMP 等。一个程序可以分解成一系列小任务块，每个小任务块包括各自

的一系列指令。互相不依赖的一组小任务块，即可考虑并行执行（分叉出一组并行的线程），然后通过合并（Join）同步，等所有进程结束后再往下运行

39. 源到源的转化（多文件到多个流文件）
40. 在 MPI 应用程序中，同一程序的多个副本执行为分离进程，通常位于计算集群的不同节点上。
41. 分布式，并行访问 HDF5 文件，每个计算节点访问不同得部分
42. 支持并行构建(利用 CPU 的额外内核)
43. 允许多线程并行操作，能够编写单个程序内置于三个不同的可执行文件中，用于不同的单节点

2. 将用户的类 C 程序（也可以看作是并行）转换成下列形式的并行程序

⑥ OpenMP
⑦ OpenMp+SIMD
⑧ Cuda
⑨ ACC
⑩ 串行的程序

(以上五点均支持 MPI)