第五次作业

程远 2234412848

1 第一题

1.1 启示

利用不同硬件间的特性来进行针对性的计算，例如写程序时，特别是在进行高性能计算时，应根据目标硬件的特点来选择合适的算法和优化策略。对于多核处理器，尽量并行化任务，对于 GPU 或者利用 CUDA 加速的程序，尽量利用其并行计算能力；

并行化是提升程序性能的关键手段之一。通过指令级并行、任务级并行和数据级并行，可以充分利用 CPU、GPU 的多核、多线程特性来大大加快程序运行速度；

提高时钟速度和发热能耗间存在 trade-off，不能过于依赖高时钟周期带来的高效运行，而应该通过优化结构，减少不必要的内存访问和读写，在不增加功耗的同时提高性能；

程序性能的瓶颈可能出现在意想不到的地方，需要我们持续进行性能分析和优化。使用性能分析工具（如 gprof、perf、Intel VTune）定期检查程序性能，找出瓶颈并进行优化。对热点代码进行详细分析，优化其执行效率；

对于并行计算，若仅对计算机一部分做性能提速，则改进越多，所得到的总性能提升越有限。例如一半的程序不能并行执行，这时用 64 核并行，只会加速 2 倍左右，但却花费了 64 倍的功耗。所以不能过于依赖并行优化，应该把算力合理分配到不同的优化方案上，或者仅加速经常发生的事件，如储存系统与指令集系统。

1.2 如何写出更快代码

一、指令级并行通过流水线技术和超标量架构来提高指令级并行度，但需要处理好分支预测和流水线控制问题。编写代码时，尽量减少分支和跳转，保持流水线的顺畅运行。无论分支多少，都可以使用条件指令或预测分支来减少流水线停顿。还可以使用编译器优化选项，允许编译器重排指令顺序以提高超标量处理器的利用率。通过算法和代码结构优化，尽量提高分支预测的准确性。使用探测器工具分析分支行为，调整代码结构以减少错误预测的频率。

二、任务级并行将任务分解为多个子任务，通过多核或多 CPU 并行处理可以有效提升性能，但并行化的潜力受限于任务的独立性和 Amdahl 定律。优化策略大致如下。先将大任务分解为多个小任务，确保这些任务尽量独立，减少任务之间的同步和通信开销。在多核处理器上，使用多线程或多进程对这些小任务进行并行处理。利用并行框架（如 OpenMP、TBB）简化并行编程。在任务级并行中，我们需要设计合理的任务调度算法，确保各处理器核心的负载均衡，避免某些核心过载而其他核心空闲。根据 Amdahl 定律，还可以尽量减少无法并行化的部分，提高并行代码的执行比例。

三、数据级并行将数据分配到不同处理器上并行处理，例如 SIMD、架构，可以显著提高计算密集型任务的性能。优化策略大致如下。先使用 SIMD 指令集进行向量化操作。使用支持数据并行的库和框架，如 CUDA，实现数据并行处理。确保数据在内存中的对齐，以充分利用 SIMD 指令集。使用编译器提供的对齐指令和编译选项。对于大规模数据处理任务，使用批量处理技术，一次性处理大量数据，减少内存访问开销。

四、硬件加速器与专用架构使用领域特定架构（DSA）如 GPU、FPGA、AI 加速器等，可以针对特定计算任务大幅提升性能。优化策略大致分为 GPU 加速、FPGA 加速、AI 加速和协同加速。GPU

加速：在计算密集型任务中使用 GPU，通过 CUDA 编写并行代码。特别适用于深度学习、图像处理等计算场景。FPGA 加速：对于需要低延迟、高吞吐量的应用，使用 FPGA 进行硬件加速。编写专用的硬件描述语言代码，实现定制化硬件加速器。AI 加速器：在机器学习任务中，使用 AI 专用加速器（如 TPU、NPU）来提高训练和推理速度。协同计算：结合使用 CPU、GPU、FPGA 等多种硬件加速器，发挥各自优势，通过协同计算实现更高性能。

2 第二题

ISA——是 Instruction Set Architecture（指令集体系结构）的缩写，指令集因其系统性和复杂性，也被称为（Instruction Set Architecture，ISA）。指令集对上限定了软件的基本功能，对下制订了硬件实现的功能目标，因此指令系统的设计（指令集中应该包含哪些指令，指令应该采用什么样的格式来表示）是计算机系统设计中十分重要的一环。

ISA 可以分为 CISC 和 RISC 两种，CISC 是指复杂指令系统计算机（Complex Instruction Set Computer），RISC 是指精简指令系统计算机（Reduced Instruction Set Computer）。CISC 的特点是具有大量复杂的指令，也得益于此，CISC 架构下的代码长度一般较短。虽然复杂指令减少了编写代码的工作量，但是也增加了 CPU 设计和实现的难度，例如 Inter 酷睿系列处理器均采用 x86 架构，由于复杂的指令与寻址，导致其功耗较高，发热严重（相对于 ARM 芯片）。RISC 的特点是精简与定长指令，如 ARM 架构。这种架构的指令简单，执行速度快，但需要更多的指令来完成复杂操作，可能增加程序长度。在功耗方面，RISC 指令集的芯片功耗往往相对于 CISC，例如 Apple 的 M 系列芯片和绝大多数手机芯片，其极低的功耗便部分得益于采用了 ARM 架构。

ISA 规定了处理器可以执行的指令、操作方式、访问内存的方式、寄存器使用方式，直接影响处理器性能。下面借助 CISC 和 RISC 的区别，详细分析 ISA 在硬件功能上的影响。指令集层面，CISC 架构有复杂的指令集，单条指令可能执行多个操作，复杂的指令解码和执行逻辑会增加处理器设计的复杂度，可能降低时钟速度并增加功耗。架构采用精简的指令集，指令解码和执行逻辑简单，高效的流水线设计使得处理器能够以更高的时钟速度运行，同时降低功耗，提高性能。指令执行效率层面，RISC 架构的指令长度固定且指令集简单，使得指令级并行（如流水线和超标量架构）的实现更为容易。流水线设计能够在每个时钟周期执行多条指令，提高处理器的吞吐量。CISC 架构的复杂指令可能需要多个时钟周期才能完成，这限制了指令级并行的有效利用，降低了执行效率。寻址与内存层面，ISA 定义了指令如何访问内存，包括直接寻址、间接寻址、基址加偏移寻址等。复杂的寻址模式会增加指令解码和执行的时间，影响处理器性能。RISC 架构通常采用简单的寻址模式，优化了内存访问的速度，提高了数据加载和存储的效率。寄存器层面，ISA 定义了寄存器的数量和用途。更多的寄存器意味着更多的数据可以在寄存器中保存，减少对内存的访问频率，提高性能。RISC 架构通常提供更多的通用寄存器，这有助于减少内存访问，提高程序运行速度。而 CISC 架构的寄存器数量通常较少，需要频繁的内存访问，影响性能。并行计算层面，ISA 可以支持并行计算，通过 SIMD（单指令多数据）或 MIMD（多指令多数据）架构实现数据级并行和任务级并行，提高处理器的并行计算能力和性能。RISC 架构由于指令集简单且统一，通常更容易实现向量化操作和并行计算，提高数据处理效率。

3 第三题

程序首先定义了一个生成随机数的函数和一个找到数组最大值的函数，过于简单，不多赘述。之后在 main() 中，程序用 fork() 创建子进程 1，其位于 else if (pid == 0) 代码块中。这个代码块下方的 else 代码块为父进程，在这个父进程中再次调用 fork() 创建子进程 2，其位于 else if (pid2 == 0) 代码块中。子进程 1、2 分别找到索引为 0 至 4999、5000 至 9999 的数组的最大值，并打印结果。父进程通过调用两次 wait(NULL) 等待两个子进程执行完毕。