Introduction to Scientific Computing (BSR1015) Homework #2

Assigned January 31st, Due February 14th

1. Why are computer chips not getting physically “faster”? What are we doing these days to get more performance per chip? What are we doing these days to get more work done over all?

* The chips are not getting faster in a few years due to the limitation of Moore’s Law. If the number of transistors in integrated circuit doubled in size every two years, the space in a single chip will reach the limit to accommodate further increased transistors. As limited amounts of circuits can be fit into a chip, parallel computing that relies on the growth of transistor counts for heat dissipation will not be facilitated in a single chip.
* To increase performance per chip is by optimization in parallelizing multiple processors onto a chip. This method distributes the tasks to each processor, which has access to shared memory and can executes a instruction on the same or different data than other cores (ch 5 slides). However, performance of per chip is limited by Moore’s law.
* To get more work done overall, we can rely on parallel computing over a network (cluster). Cluster is not restrained by Moore’s Law due to its utilization of connecting multiple computers that have multiple processors, in which the transistor counts are not limited by the size of a single chip. This approach speeds up processing by breaking down the task into pieces for each node to work on, and hundreds of nodes work together and create the high-speed network, which is beneficial to solve the problem. The Batch system, utilized by the classic HPC cluster, processes thousands of jobs, and it scales well to 1M+ processors linearly. Performance of HPC is comparable to 250 Teraflops aggregate. More work can be done by job scheduling—creating jobs to accomplish each task.

1. In class, we scaled up our knowledge of parallelization from single threaded problems through SIMD acceleration, multicore and finally distributed computing across a network. What common theme unites all of these strategies? (Hint: think about the problem requirements to be compatible with parallel computing).

The problem requirements compatible with parallel computing include massive computation or processing power. The goal of parallel computing is to increase computing power by breaking down a task into subtasks, and the processors in a chip or in different nodes can simultaneously execute the subtasks. The concept resembling parallelization has started from the design of SIMD that aims to process multiple data with a single instruction, which is different from conventional sequential approach that one instruction processes each individual data. Processing with multicore further actualizes parallel computing by running instructions on multiple cores in a single processor. The ultimate approach of parallel computing is to connect multiple processers in a node and the combined nodes can run on different tasks.

1. On a modern UNIX/Linux computer, how many single-precision floating point numbers can fit into an SSE or AVX-128 register for SIMD acceleration?

Lecture Slide 15 of ch5 suggests 4 single precision floats can be fit into a 128-bit wide vector registers.

1. Instead of writing our own SIMD code using vector intrinsics, how can we get a computer to automatically accelerate our code using these features? What are the possible downsides to automatic SIMD acceleration?

Compiler optimizations [1] can automatically accelerate coding than parallelization and vectorization. For automatic-parallelization, the complier detects the code in the loop that is potential to be parallelized and converts the scalar operations into parallel operations (an algorithm). For automatic-vectorization, The compiler (automatic vectorizer) uses vector registers [3] and packed SIMD instructions on more than one data element [2] to execute loops in your code, so the complier can automatically identifies and optimizes suitable loops more efficiently without manual actions. The information of whether a loop was vectorized can be accessed by the optimization report. The possible downsides to automatic SIMD acceleration is sometimes something not relevant such as keywords or directives [2] could be accidentally written into the algorithms by the complier.

Sources: [1] Lecture slides ch5

[2] Intel: <https://software.intel.com/en-us/node/522572>

[3] Microsoft: https://msdn.microsoft.com/en-us/library/hh872235.aspx

1. Describe a science problem that you may use a supercomputer/cluster to solve. (It can be any science problem, real or fake. Does not have to be related to your field.) Why do you think this problem scales to a supercomputer/cluster? Why would it not be optimal to run this problem on a desktop/laptop?

A science problem such as mapping human brain function relies on a supercomputer rather than a regular PC, because this works requires mapping of 31,000 virtual brain cells connected by roughly 37 million synapses [1]. The computer power required to model neural activity in each level of molecules and circuitry would be roughly about an exaflop, or 1018 operations per second [2]. This kind of work should be split into multiple jobs for the latest supercomputers that usually measure teraflops (1012) to speed up the modeling process. Not even an old supercomputer like Cray-1 that could just do 160 megaflops was optimal enough. Some of the modern desktop processor like Intel i7 920 that scored about 63 gigaflops may be more compatible than the old supercomputers [2]. Also, Intel seemed to be interested to produce a new chip that runs a teraflop on a desktop [3]. Nevertheless, Nvidia, the GPUs manufacture is planning to scale up performance of GPUs (for supercomputers) to exaflop by 2021 [4]. Back to Moore’s law, performance of parallel computing in CPUs will not surpass GPUs after a certain limit is reached.

Sources: [1] Fragment of rat brain simulated in supercomputer, Nature

[2] Computer modelling: Brain in a box, Nature

[3] Intel’s new chip puts a teraflop in your desktop. Here's what that means, Popular Science

[4] Update on the race to the Exaflop supercomputer, Next Big Future

1. How would you make batch jobs for simple existing programs on Minerva? What would you do to run your work in a job? Give an example of a job you would create and what parameters you might provide.

* To make batch jobs programs on Minerva, we first run a scheduler (LSF) to share computers among users, and the resources for the jobs will be allocated to the Batch system. The scheduler will specify the number of cores, the name of application and how long required to run the jobs. Then, we created jobs by submitting batch scripts to the batch system. Despite the default setting for batch parameters, we are better to request queue, the number of cores needed, resources and wall time for the job, depending on how big the job is. LSF will assign the number of cores requested to the created batch job. We can execute the jobs with the accessed resources.
* An example of a job to create is to make a file named “goodjob.sh”, which will be stored in my home directory as “openmp+opt”. I enter the parameters as follows by writing to “nano goodjob.sh” and execute a.out. We need to save the file before job submission. I submit my job by piping the file as stdin, through “bsub”, which I type “bsub <goodjob.sh” and enter “bsub”. I check whether “goodjob.sh stderr.N.0” and “goodjob.sh stdout.N.0” are in the same current directory with “ls” and use “cat stdout.N.0” to see my submitted parameters, and the information about how it was scheduled.

1. #!/bin/bash
2. #BSUB -P *acc\_BSR101*
3. #BSUB -q expressalloc
4. #BSUB -n 48
5. #BSUB -R span[ptile=12]
6. #BSUB -R rusage[mem=12000]
7. #BSUB -W 02:00
8. #BSUB -o %J.stdout
9. #BSUB -eo %J.stderr
10. In class, you noticed that the time required for vector-vector addition did not continue to decrease beyond about 10 parallel tasks. The figure is replicated below. Describe why this might be true. (Hint: what other resources are finite in a modern computer beyond CPU speed?).

PING-MASTER:Users:abcosta:Documents:sinai:class:figures:plot_parallel.pdf

We don’t usually get N times of [speedup](https://en.wikipedia.org/wiki/Speedup) when running a program on the N processor platform by OpenMP due to the main reasons as follows. A process sometimes has to wait until the data that depend on to be computed to proceed [1]. For example, a thread must waits until the other threads to release resources, when multiple processes share non-parallel resources [1]. Pertaining to Amdahl's Law, [speedup](https://en.wikipedia.org/wiki/Speedup) of task execution of a task at the fixed [workload](https://en.wikipedia.org/wiki/Workload) is limited in a system where the resources improve [3], which is due to the reason that parts of a code are inherently sequential. The speedup is limited by a factor. In a code, we suppose Fs=sequential fraction, and Fp=parallel fraction. The parallel execution time on processors is the sum of the part that is sequential T1Fs and the part that is parallel T1Fp/P as this equation: TP=T1(Fs+Fp/P). As the number of processors increases (P→∞), the time of parallel execution decreases and reaches to that of the sequential fraction of the code and the speedup is limited [2]. If a particular part within a program that takes 1 hour of execution cannot be parallelized, the time of execution the whole program will not be minimized than that 1 hour disregarding the number of hours capable to be parallelized [3].

Sources: [1] OpenMP, Wikipeida

[2] http://pages.tacc.utexas.edu/~eijkhout/istc/html/parallel.html

[3] Amdahl's law, Wikipedia

1. In class, we discussed how accelerator devices such as GPUs and Intel MICs can be orders of magnitude faster than traditional CPUs for certain types of calculations. However, they are not candidates for replacing CPUs for general purpose computing. Why not? (Hint: do some internet research).

GPUs and Intel MICs benefit certain types of calculations, i.e. highly parallelizing processing and high-performance computing (HPC). GPUs are often compared as an accelerator for computational calculations. The fundamental differences in architecture and techniques to process programs between GPUs and CPUs make both of them excel at different functions. GPUs spread their workload across many threads, but each thread is not as efficient as a single thread in the CPUs. If it’s not for the purpose of speedy calculation, using GPUs alone would not be worthy by dramatically increasing the power requirements due to their complex processing on simple data [1]. These features make GPUs specialized for image processing. They are also good at physics simulation in video games and at accelerating video encoding and decoding [2]. However, GPUs do not have the features of interrupts and virtual memory that are important to implement a modern operating system [2]. Jobs like running an accounting package or Microsoft Word on the operating systems cannot be compatible with GPUs [1]. For general purpose computing, CPUs are more compatible than GPUs, for each core of CPUs runs faster than GPUs in terms of instructions per second. They handle single or lower amounts of data more efficiently than GPUs.

Sources: [1] GPUs aren’t going to replace CPUs, but they are here to stay, scientific computing world

[2] [Why are we still using CPUs instead of GPUs?](https://superuser.com/questions/308771/why-are-we-still-using-cpus-instead-of-gpus), StackExchange

1. After you decide what your computational strategy is for a certain problem, what is the first thing you should do when researching a potential solution?

Assuming I have worked on CUDA GPUs or Intel MICs for HPC, I will install a few libraries for programming in Python, R, or C++ to solve certain questions in dense linear algebra or matrix-matrix operation, as follows [1],

* LAPACK (Linear Algebra Package) – the software library for numerical linear algebra for solving linear systems of equations and singular value problems
* ScaLAPACK (or Scalable LAPACK)  – for distributed memory [MIMD](https://en.wikipedia.org/wiki/MIMD) parallel
* PLASMA (Parallel Linear Algebra Software for Multicore Architectures)  – massive Shared Parallel
* MAGMA – linear algebra GPU accelerated libraries for heterogeneous GPU-based architectures (multicore CPUs and multi-GPUs) [2].

Sources: [1] Ch5 lecture slides

[2] Nvidia

\