Productive high-performance software for OpenCL devices

John Melonakos*[a], Pavan Yalamanchili[a], Chris McClanahan[a], Umar Arshad[a], Michael Landes[a], Shivapriya Jamboti[a], Abhijit Joshi[a], Shehzan Mohammed[a], Kyle Spafford[a], Vishwanath Venugopalakrishnan[a], James Malcolm[a]
[a]AccelerEyes, 3423 Piedmont Rd NE STE 330, Atlanta, GA, USA 30305

ABSTRACT

Over the last three decades, CPUs have continued to produce large performance improvements from one generation to the next. However, CPUs have recently hit a performance wall and need parallel computing to move forward. Parallel computing over the next decade will become increasingly defined by heterogeneous computing, involving the use of accelerators in addition to CPUs to get computational tasks done. In order to use an accelerator, software changes must be made. Regular x86-based compilers cannot compile code to run on accelerators without these needed changes. The amount of software change required varies depending upon the availability of and reliance upon software tools that increase performance and productivity. Writing software that leverages the best parallel computing hardware, adapts well to the rapid pace of hardware updates, and minimizes developer muscle is the industry's goal. OpenCL is the standard around which developers are able to achieve parallel performance. OpenCL itself is too difficult to program to receive general adoptions, but productive high-performing software libraries are becoming increasingly popular and capable in delivering lasting value to user applications.

Keywords: GPU, OpenCL, CUDA, Xeon Phi, FirePro

1. BACKGROUND ON CPU PROCESSING TRENDS

Over the last three decades, CPUs have continued to produce large performance improvements from one generation to the next. However, CPUs have recently hit a performance wall and need parallel computing to move forward.

1.1 Clock frequency improvements for CPUs

Up until 10 years ago or so, CPUs improved in speed mainly by increasing in the frequency of their clocks. For instance CPUs progressed from MHz speeds to 1 GHz to 2 GHz to 3 GHz and some even push into 4 GHz ranges. However, it is rare to see CPUs running above 4 GHz. There is an important reason for this. If the frequency gets too high, the chip can actually melt from the excessive power/heat. Today CPUs are typically seen running at 2 to 3 GHz, which is where they landed over 10 years ago.

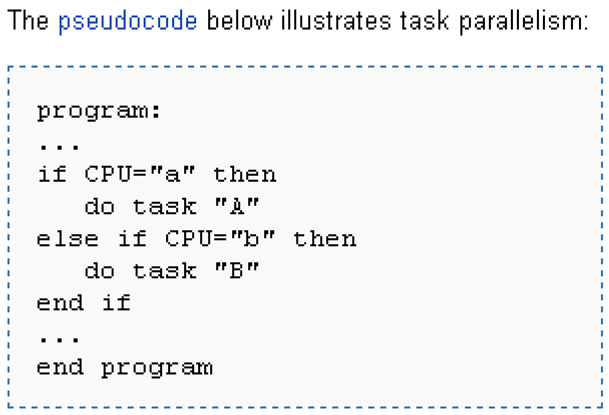
There are ways to make CPUs run faster. But that requires keeping them cool so they don't melt. Even at 2-3 GHz a computer needs a lot of fans and heat conducting metal apparatuses (called heat sinks) to whisk away the heat from the chip. That's why computers make noise; it's the fans keeping the CPU cool. But fans only work up to 2-3 GHz. To push above that frequency range, liquid cooled solutions are needed. However, short of really hardcore computer geeks and gamers, no one really wants to put liquid in their computer.

1.2 Task parallelism on CPUs

So, in order to keep improving speed over the last 10 years, CPU manufacturers had to find other ways to improve things. They found improvements in adding more CPU cores (each of which is a full CPU) onto the same CPU chip. One of the cores could do one thing (like play a movie) which the other core did something else (like run Microsoft Excel). Since those tasks were split up between different CPU cores, each core did not have to work as hard to get the tasks done and faster experience could effectively be had without raising the clock frequency.

This is where parallel computing first entered the technology scene in a big way. On CPUs, this is called "Task Parallelism." In order for programs to actually use all the CPU resources, the software had to be re-written and re-compiled with special consideration for the fact that the processor architecture had changed.

Modeling and Simulation for Defense Systems and Applications VIII, edited by Eric J. Kelmelis,
Proc. of SPIE Vol. 8752, 87520C · © 2013 SPIE · CCC code: 0277-786X/13/$18
doi: 10.1117/12.2016216



The pseudocode below illustrates task parallelism:

```
program:
...
if CPU="a" then
   do task "A"
else if CPU="b" then
   do task "B"
end if
...
end program
```

Figure 1. Pseudocode of task parallelism from wikipedia.org.

1.3 Limits of multi-core processors

So for the last 10 years, the industry has moved from single core, to dual core, to quad core. But again, it is uncommon to see CPUs with more than 4 actual cores. That is because with 4 cores on one chip, the size of the chip grows and the power/heat starts to rise again. So again, CPUs have hit again hit some physical barriers. And this time CPUs are having a much harder time figuring out how to deliver substantial performance improvements. Mainly CPUs are relying on making transistors smaller, so that the power per core requirements go down, so that more cores can be added.

1.4 Good enough?

Luckily, for most people, regular dual or quad core CPUs are good enough for the tasks that need to be done. If a computer is slow, it likely is not due to the CPU anymore. The bottleneck is much more likely to be the hard drive, upgrading to an SSD is the best way to speed up a standard computer today.

However, for scientists, engineers, and financial analysts (i.e. for people that run big simulations), CPUs are still slow. AccelerEyes was founded in 2007 on the cusp of a transformative computing event when the high-end computing professionals realized CPUs were no longer going to improve as fast as is needed for their applications.

1.5 The rise of heterogeneous computing

The answer to this problem was found in using other processors to supplement the CPU in getting the computing work done. It started with leveraging the GPU (graphics processing unit) on the video card as a companion to the CPU in computational tasks.

In my next post, I'll talk more about how GPUs have made a permanent home in the world of computing and how heterogeneous computing is the name of the computing game for the next decade.

1.6 Additional notes

CPUs have also improved greatly over the years in architecture, caches, prediction, and more. But those improvements have not been sufficient to stem the tide of heterogeneous computing.

CPUs also have ways to offload computations to data-parallel sections of their same chip (using SSE/AVX instructions). Those options are also not significant enough to alter the macro-level trends.

2. BACKGROUND ON HETEROGENEOUS COMPUTING

Over the last 20 years, big gains in computer processing have been defined by increases in CPU clock speeds, then by increases in the number of CPU cores. The next 10+ years will be defined by heterogeneous computing.

2.1 Heterogeneous computing

It is helpful to begin with a definition: Heterogeneous computing is the coordination of 2 or more different processors, of different architecture types, to perform a computational task. "Architecture type" is defined below.

In practice, that means that there is actually more than one processor in the computer. The general purpose CPU that is common place (of x86 architecture type) is typically present, but also another processor (of a different architecture type) is present in the system. That other processor is an "accelerator," because it accelerates computations by assisting the CPU to get stuff done.

Most computers already have an accelerator as a component. Many computers have more than one accelerator already present, even if the computer's owner did not seek out the accelerator specifically for computational purposes. Some of those accelerators may even be more powerful in terms of ability to do processing than the CPU. However, it is very likely that the computer's software is not using those accelerators for acceleration purposes. The accelerators are probably just sitting there unused.

This is described in more detail below.

2.2 The rise of GPU computing

In most computers, there is a GPU (graphics processing unit), which is an accelerator to the CPU. Most people are aware of the GPU (which typically resides on a video card) as the thing that drives the computer monitor. The computer monitor is plugged into the video card/GPU.

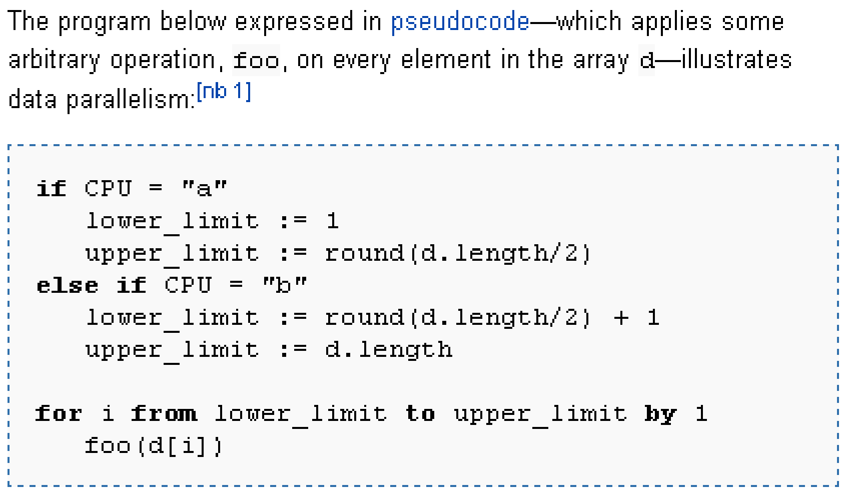
Innovations in GPUs over the last 20 years have been primarily driven by the demand for more awesome video games. Video games have advanced tremendously, going from Mario Brothers (which simply had some pre-computed pixel patters that are the same every time they are played) to the games today which are extraordinarily complex in how they perform physics calculations of wind-blowing, water-flowing, trees-swaying, and all sorts of other physical phenomenon. In games, GPUs can actually do all those physics calculations on-the-fly to determine the color of pixels to be sent to the monitor. It still blows my mind that there is a company[1] purely dedicated to creating physical models for trees in video games (i.e. those physics calculations are extremely complex and the GPU has to be a beast of a processor to handle them).

In order to support all of those physics calculations, GPUs have advanced from merely displaying to the monitor to actually having incredible capabilities to do math computations. The processing capability of the GPU is not limited to video games. In fact, any software program can use the GPU to do computations. Complex math can be executed on GPUs. Financial calculations can be performed on GPUs. Genomic sequencing can be accomplished on GPUs. Radiologists can find tumors in MRI scans using GPUs.

In terms of sheer capacity to crunch numbers, GPUs can crunch more numbers per minute than CPUs. They have thousands of cores for number crunching. They are more powerful. They also use less energy per computation than CPUs. Note that a GPU core is not nearly as capable as a CPU core in terms of the kinds of things they can do, but there are many more of them available.

The ability of GPUs, and other accelerators, to perform so well has to do with their ability to leverage "Data Parallelism." The hundreds of light-weight cores on these accelerators enables them to crunch mathematical operations on many different data points simultaneously.

[1] http://www.speedtree.com/



The program below expressed in pseudocode—which applies some arbitrary operation, foo, on every element in the array d—illustrates data parallelism:[nb 1]

```
if CPU = "a"
    lower_limit := 1
    upper_limit := round(d.length/2)
else if CPU = "b"
    lower_limit := round(d.length/2) + 1
    upper_limit := d.length

for i from lower_limit to upper_limit by 1
    foo(d[i])
```

Figure 2. Pseudocode of data parallelism from wikipedia.org.

GPUs are also ubiquitous. Every computer, smartphone, and tablet has GPUs in them.

2.3 Software for heterogeneous computing

It is very fortuitous for the computing industry that these computational powerhouse accelerators are in so many computers already.

However a serious challenge is presented in enabling software to run on the accelerators. It is not so fortuitous that most software is incapable of actually running on those accelerators. In order to use accelerators, software must be re-written.

The process for re-writing software involves an understanding of parallelism. Software written for parallel computing runs circles around other software, because it is able to use the multiple cores of the CPU as well as the many cores of accelerators, like GPUs.

2.4 The next decade: a tidal wave of heterogeneous computing

The rise and success of GPU computing over the last 5 years, with NVIDIA as the hardware-vendor leader, solidified the validity of accelerators and has provoked a tidal wave of oncoming heterogeneous computing systems. As followers to NVIDIA, here is a list of other companies and their accelerators pushing heterogeneous computing as the primary path to computational performance increases over the coming decade:

- Intel Xeon Phi – released this year as a 60+ core accelerator (they prefer the term "co-processor"). It has a new processor architecture with a ring of older x86 CPUs.
- Intel integrated graphics – this is Intel's GPU. It is not as capable as NVIDIA or AMD's GPUs, but comes on the same chip as all Intel's CPUs and is probably the most ubiquitous GPU today for that reason.
- AMD FirePro and Radeon – these are the only other first-rate GPUs for desktops and servers.
- AMD APUs – this is AMD's merger of Radeon technology onto the same chip as the AMD CPU (i.e. APU = CPU + GPU). The GPU AMD is putting on APUs today is not as powerful as the full Radeon GPU, but it can still be used as an accelerator.
- Altera FPGAs – these used to be restricted to very niche markets, but with the tidal wave towards heterogeneous computing, Altera's FPGAs and FPGAs from other vendors will be considered as a viable option for many more applications.

In addition to those, all smartphones and tablets have CPUs and GPUs. All these heterogeneous computing concepts apply equally well to getting more performance out of mobile apps.

With all of these major companies pushing accelerators, heterogeneous computing is easily the biggest trend in computing for the coming decade.

2.5 Additional notes

"Architecture type" refers to the blueprint used to create the processor. Software that will run on one architecture will not run on another architecture without modification, either of the code itself or with the way it was compiled to run. AMD and Intel use the same blueprint, called "x86″, so software compiled for AMD will generally run on Intel and vice versa. Heterogeneous computing involves many different processors of different architecture types. Developing software for all those different types is complicated and will be described in a future post in this series.

When we have described heterogeneous computing to people in the past, they often get over excited about accelerators and ask, "When will accelerators overtake CPUs and when will we no longer need CPUs?" That question misses the point. A good analogy is to think of a computer as an army. In that paradigm, CPUs would be the generals – highly capable and extremely efficient at command and control. Accelerators would be the foot soldiers, massive numbers of production units but not as capable at decision-making.

3. PARALLEL SOFTWARE DEVELOPMENT AND OPENCL

As described above, in order to use an accelerator, software changes must be made. Regular x86-based compilers cannot compile code to run on accelerators without these needed changes. The amount of software change required varies depending upon the availability of and reliance upon software tools that increase performance and productivity.

There are four possible approaches to take advantage of accelerators in heterogeneous computing environments: do-it-yourself, use compilers, use libraries, or use accelerated applications (if lucky).

3.1 Do-it-yourself

The do-it-yourself approach is defined by taking an inordinate amount of software development time to write code that leverages all the parallel attributes of your heterogeneous computer. Hardware vendors such as AMD, Intel, and NVIDIA provide access to low-level tools that enable developing parallel code to run on their heterogeneous hardware.

NVIDIA is the leader with the CUDA platform. CUDA is the first highly adopted platform enabling high-performance general-purpose GPU code. Developers of CUDA code write GPU kernels, manage different levels of GPU memory, make tradeoffs for data transfers between the host and the device, and optimize many other aspects of the parallel system.

OpenCL is the industry's open standard for similarly writing data-parallel code in heterogeneous computers. AMD and Intel both promote OpenCL as a primary approach towards programming their parallel computing hardware offerings. OpenCL requires a similar level of low-level understanding and competence to write efficient parallel software.

Both CUDA and OpenCL are primarily targeted at leveraging data-parallelism of devices, and additional considerations must be made to use the multiple cores available on the CPUs in the system. For instance, OpenMP and MPI enable the use of multiple CPU cores and compute nodes in a heterogeneous system.

The do-it-yourself approach is very costly in terms of developer muscle and expertise. The do-it-yourself approach does not adapt well to ongoing hardware updates. For instance, a parallel computing algorithm tuned for NVIDIA's Fermi GPUs may not be the best algorithm choice for Kepler GPUs. These difficulties are what prompted AMD to say that most programmers will not use CUDA or OpenCL[2]. We agree. Most programmers will be smarter and will use the more productive approaches described below.

The best attribute of the do-it-yourself approach is that it's always available in case other more productive approaches fail.

3.2 Use compilers

Serious research attempts have been made by compiler developers to offload the brunt of parallel software development from manpower to compiler power. Unfortunately, these research attempts have not been very successful and the task of automatically finding the parallelism in a code is an unsolved research problem.

[2] http://www.theinquirer.net/inquirer/news/2257035/amd-thinks-most-programmers-will-not-use-cuda-or-opencl

There are some use cases where simple, straightforward loops can be unrolled by compilers and executed efficiently on parallel hardware without much human intervention. But those are not commonplace.

Compilers can also get smarter as software developers introduce compiler-targeted directives (or software changes) that give the compilers hints about which things should be parallelized. However, many people have noticed that by the time you add sufficient compiler hints to the code to get good performance, you often would have been better off simply going the do-it-yourself approach in the first place.

Compilers will continue to improve, but for many decades theoretical researchers have tried to solve the problem of automatic parallelism detection from code and have failed. In fact, it has been proven that it is impossible to automatically detect parallelism in many instances.

The best attribute of compilers is that they work on simple arithmetic in loops; so if you have a very simple use case with simple operations, compilers are worth consideration.

3.3 Use libraries

Libraries benefit from the great performance of the do-it-yourself approach, as well as the easy-of-use of the compiler approach. Software libraries are written by parallel computing experts who are focused on optimizing the last bit of advantage out of one narrow function at a time.

Today, heterogeneous software libraries are built on top of the CUDA or OpenCL platform and selection of the appropriate library depends upon the choice of platform. There are more CUDA libraries available today than OpenCL libraries, largely because NVIDIA has done a marvelous job at building a parallel computing ecosystem. However, with the recent emergence of Intel Xeon Phi and the growth of OpenCL's utility in mobile computing, OpenCL libraries are becoming more and more prevalent than before.

Use of libraries requires a trust upon the library's developers. Common qualifying questions regarding libraries include: Is the library fast? Is the library stable? Is the library fully supported? Will the library be updated quickly as new hardware updates occur? Is there a community built around the library?

Libraries that have strong responses to those questions have great benefits for software developers. Libraries that do not can be a waste of time, and developers would do well to not waste their time pushing on broken software.

The best attribute of libraries is the ability to leverage expertly written parallel software while avoiding the time-sink of the do-it-yourself approach.

3.4 Use accelerated applications

As heterogeneous computing becomes more prevalent, many popular applications are becoming accelerator-enabled already. For instance, Adobe has products that are CUDA and OpenCL-accelerated for faster video transcoding. We are working with MathWorks on parallel tools. Ansys has GPU-accelerated products. And there are many more.

Users of those products do not have to do heavy-lifting to get faster code. They simply get to benefit from the work done by the application developers, often with simple checkboxes or designations that indicate a preference towards accelerated computing.

3.5 The next decade: challenges for parallel software

The next decade will be defined by how the industry responds to the challenges of developing parallel software for heterogeneous computers, from high-performance computers down to mobile devices. Poor choices will lead to outcomes like that of the Roadrunner[3] supercomputer (and IBM cell processor) which is being decommissioned after only a few years of use (see note below). Writing software that leverages the best parallel computing hardware, adapts well to the rapid pace of hardware updates, and minimizes developer muscle is the industry's goal.

OpenCL is the standard around which developers are able to achieve parallel performance. OpenCL itself is too difficult to program to receive general adoptions, but productive high-performing software libraries are becoming increasingly popular and capable in delivering lasting value to user applications.

[3] http://www.hpcwire.com/hpcwire/2013-04-04/revelations_on_roadrunner_s_retirement.html