Five Big Insights from the Student Winner of the Intel® Modern Code Developer Challenge

Submitted by Mathieu Gravey (https://software.intel.com/en-us/user/1232404) on March 22, 2016

As Shared by Mathieu Gravey, Grand-Prize Winner of the Intel Modern Code Developer Challenge

The Modern Code Developer Challenge 2015 engaged nearly 2,000 students across 130 universities and 19 countries. Grand-prize winner Mathieu Gravey, a PhD student at University of Lausanne in Switzerland, submitted the fastest optimized code and won a nine-week internship at CERN openlab.

In a recent conversation, Mathieu discussed the approach and philosophy he used to optimize the brain simulation code using modern code and parallel computing on Intel® Xeon Phi™ coprocessors. Apply these tips and techniques to your code optimization efforts and see what application processing improvements you can reap for your high-performance computing projects.

Mathieu's Key Insights

1. Embrace the unknown.

Accept that unfamiliar code can be perplexing. While it's usually easier if you can engage the developer who created the code you're optimizing, it's not always possible. And documentation to help guide one's understanding of the code can be sporadic, if it exists at all. In the Modern Code Developer Challenge, Mathieu first studied the existing brain simulation code to understand what it did. To do that, Mathieu used a bottom-up approach to understand locally what each part was doing. He was then able to create an approach to optimize it and apply the localized knowledge on a global scale.

2. Start small.

Perfect a small section of code, then apply it to the larger set. Mathieu first analyzed the speed of the data layouts—some were slow, others very fast. He first worked to improve a small layout, then applied those changes to other larger data layouts. He applied the same approach to improve functions, testing simple functions before attacking larger, longer, and more complex ones. While you may find that the application's performance from small to large data sets isn't exactly the same, in this case, the thread parallelization remained similar from the small data set to the big, creating an efficient way to speed processing.

3. Tackle big rocks first.

Starting small doesn't mean thinking small. Solve the challenging problem at the beginning. Mathieu tackled the data structure first because he knew it would be complicated to return to and change later, in part because data should be well packed and aligned to take advantage of vector operations to improve speed and efficiency. For example, changes in code that includes many pointers, like arrays of arrays or lists, may result in alignment issues with the memory. It's important, then, to make sure you're optimizing memory access and maximizing the use of cache.

4. Optimize. Test. Repeat.

Optimize, and then check again. Is there a better algorithm? How about an easier way to parallelize the code? With each iteration, make sure you validate your results for correctness. And see if using vectorization or inlining offers additional improvements. (Remember to take advantage of the compiler tool's ability to review for vectorization and inline expansion, too.) Prioritizing the optimizations may also help: that is, saving code that may be best optimized in a slightly different way to return to later. For Mathieu, that meant testing different versions of code numerous times to see if factorization or vectorization optimizations would be more effective. This transforming-and-tuning effort resulted in further incremental improvements to processing speed, and is a good approach to consider when optimizing your code.

5. Do something else. Anything else.

Cook. See a movie. Get some exercise. Mathieu completed his optimizations over the course of a month, returning to the project multiple times in between studies and daily life. Using this creative process, he brought fresh perspective and renewed energy each time.

320XFASTER

The amazing results

With this approach, Mathieu netted a striking 320x improvement in the code's runtime, from 45 hours to 8 minutes and 24 seconds. See more from Mathieu here (https://software.intel.com/en-us/videos/using-modern-code-to-simulate-brain-development-interview-mathieu-gravey).

Watch for More Insights

We'll follow Mathieu's journey as he embarks on his internship at CERN openlab later this year to work with the world's foremost research

scientists at the convergence of biology and information systems. So stay tuned for more.

Categories: Modern Code (/en-us/search/site/field_topic/modern_code-80685/language/en) , Parallel Computing (/en-us/search/site/field_topic/parallel_computing-20867/language/en) , Threading (/en-us/search/site/field_topic/threading-20869/language/en) , Vectorization (/en-us/search/site/field_topic/vectorization-35851/language/en) , OpenMP* (/en-us/search/site/field_technology/openmp-20860/language/en) , C/C++ (/en-us/search/site/field_programming_language/cc-20802/language/en) , Developers (/en-us/search/site/field_audience/developers-17152/language/en) , Professors (/en-us/search/site/field_audience/professors-17154/language/en) , Students (/en-us/search/site/field_audience/students-17155/language/en) , Linux* (/en-us/search/site/field_operating_system/linux-20787/language/en)

Add a Comment

Top

(For technical discussions visit our developer forums. For site or software product issues contact support.)

Please sign in to add a comment. Not a member? Join today ›