CASE STUDY

High-Performance Computing
Education/Scientific Research

More Efficient Numerical Simulation

intel® Software

Novosibirsk State University boosts a simulation tool's performance by 3X with Intel® Parallel Studio, Intel® Advisor, and Intel® Trace Analyzer and Collector

"The use of Intel® Advanced Vector Extensions for Intel® Xeon Phi™ processors gave us the maximum code performance compared with other architectures available on the market."

—Igor Kulikov
Assistant Professor
Novosibirsk State University

Novosibirsk State University is one of the major research and educational centers in Russia and one of the largest universities in Siberia. When researchers at the University were looking to develop and optimize a software tool for numerical simulation of magnetohydrodynamics (MHD) problems with hydrogen ionization—part of an astrophysical objects simulation (AstroPhi) project—they needed to optimize the tool's performance on Intel® Xeon Phi™ processor-based hardware. The team turned to Intel® Advisor and Intel® Trace Analyzer and Collector. This resulted in a performance speed-up of 3X, cutting the standard time for calculating one problem from one week to just two days.

Keeping Education Relevant

NSU offers about 120 programs of study in technical, economic, and humanitarian fields at the bachelors, masters, PhD, and post-doctoral levels. University staff are committed to both research and teaching and to keeping the university's educational programs relevant to the challenges of modern society. NSU works in close cooperation with research institutions and universities and with the industrial, commercial, and state sectors.

Mathematical modeling plays a key role in modern astrophysics. It is the universal tool for research of non-linear evolutionary processes in the universe. Modeling the complex astrophysical processes in high resolution takes the most powerful supercomputers. The University's AstroPhi project develops astrophysical code for massively parallel supercomputers with Intel Xeon Phi processors. This valuable project helps students learn to create numerical simulation code for massively parallel supercomputers. The students also learn about modern HPC hardware architectures—preparing them to develop tomorrow's exascale supercomputers.

Numerical Method

The team designed the project using a numerical method (Figure 1). The benefits of this high-order method included:

- The absence of artificial viscosity
- Galilean-invariant solution
- Entropy non-decrease guarantee
- Simple parallelization
- Potentially "infinite" scalability (weak scalability)

The first three benefits are the key factors for realistic modeling of all the significant physical effects in astrophysical problems. The simplicity of the method, plus the small number of MPI send/receive operations, provides efficient parallelization—and potentially "infinite" scalability in terms of weak scalability.

Novosibirsk State University
*THE REAL SCIENCE

Massively Parallel Architecture

The team co-designed the new solver for massively parallel architecture based on Intel Xeon Phi processors. Designed to help eliminate node bottlenecks and simplify code modernization, the bootable processors provided the power efficiency the team needed to handle the most demanding high-performance computing applications.

The team based the solver on Intel® Advanced Vector Extensions 512 (Intel® AVX-512) instructions, which deliver 512-bit SIMD support and enable programs to pack eight double-precision or 16 single-precision floating-point numbers, or eight 64-bit integers, or 16 32-bit integers within the 512-bit vectors. This enables processing of 2X the number of data elements that AVX/AVX2 can process with a single instruction, and 4X that of SSE.

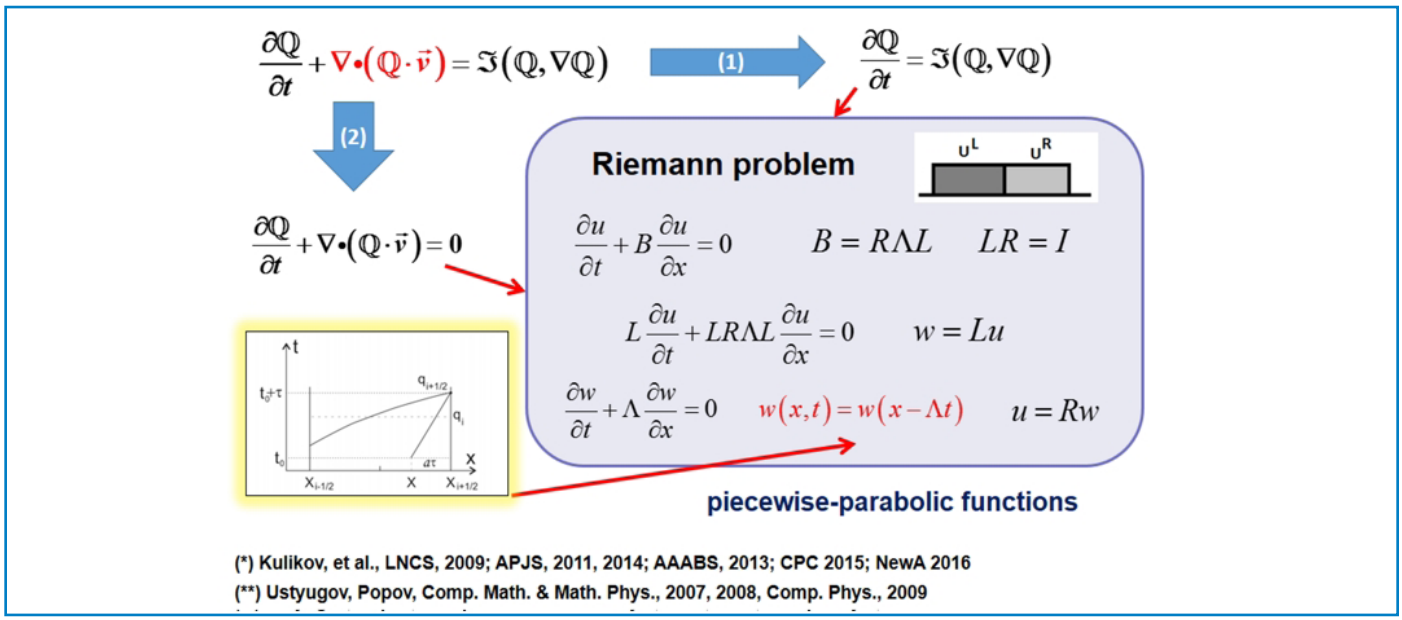

Figure 1. Numerical method

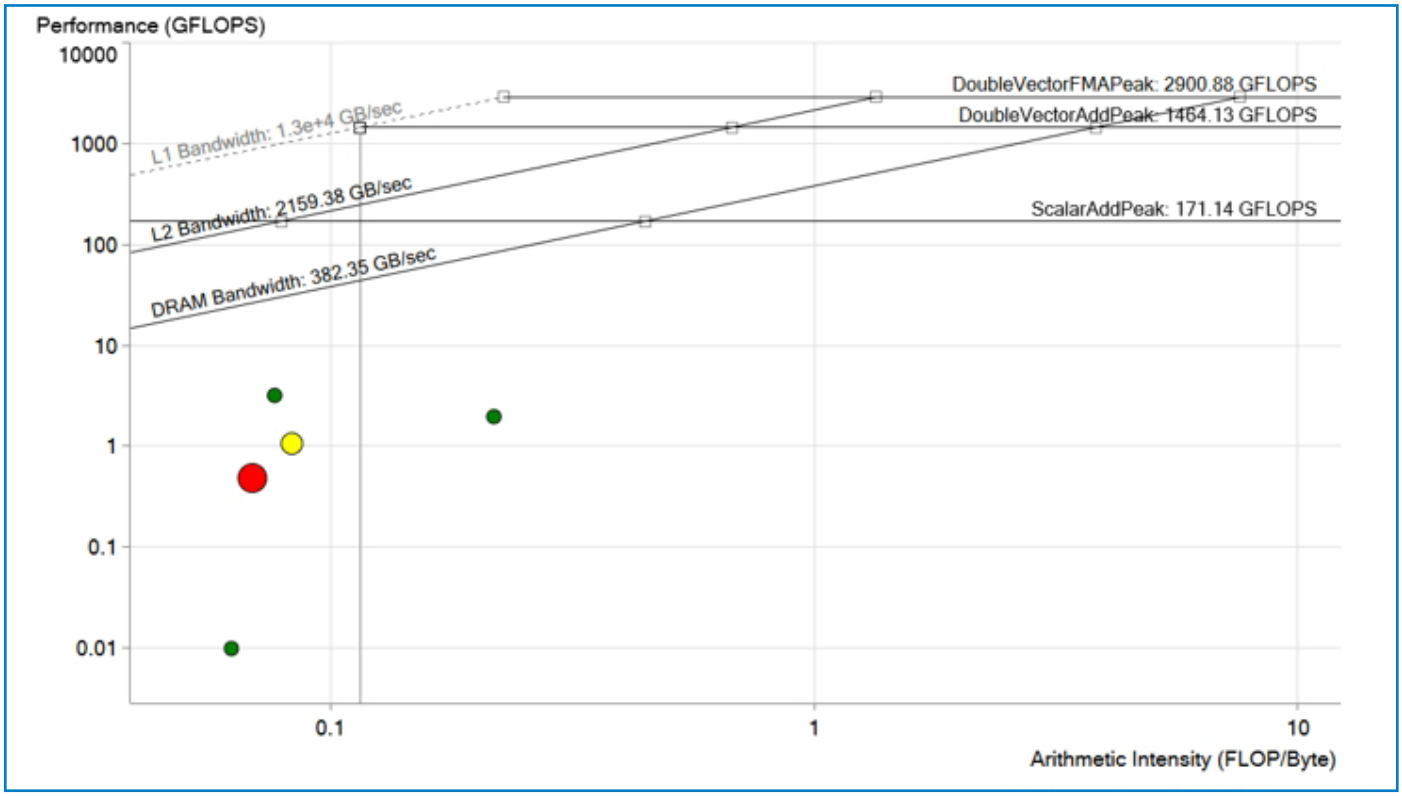

Figure 2. Before optimization. *Each dot is a loop. Bigger and redder loops take longer to run and so deliver more impact when optimized. The red loop lies well below the DRAM bandwidth peak and is running at less than 1 GFLOP. It has a lot of room for improvement and will have a big impact when optimized.*

"The use of Intel Advanced Vector Extensions 512 for Intel Xeon Phi processors gave us the maximum code performance compared with other architectures available on the market," said Igor Kulikov, assistant professor at NSU.

Optimizing the Code

A key aspect of the AstroPhi project was optimizing the code for maximum performance on the Intel Xeon Phi processors. Before optimization, the team had some problems with vector dependencies and vector sizes (Figure 2). The goals for optimizing the code were to remove vector dependencies and optimize memory load operations, efficiently adapting vector and array sizes for the Intel Xeon Phi architecture. The team used Intel Advisor and Intel Trace Analyzer and Collector, two tools that are part of Intel® Parallel Studio XE, for the optimization.

Intel Parallel Studio XE is a comprehensive software development suite that helps developers maximize application performance on today's and future processors by taking advantage of the ever-increasing processor core count and vector register width.

Intel Advisor is a software tool based on the fact that for modern processors, it is crucial to both vectorize (use AVX* or SIMD* instructions) and thread software to realize the full performance potential of the processor. Using this tool, the team was able to perform a roofline analysis highlighting poor-performing loops and showing performance headroom for each loop, identifying which can be improved and which are worth improving.

"Intel Advisor made it easier to find the cause of bottlenecks and decide on next optimization steps," explained Igor Chernykh, assistant professor at NSU. "It provided data to help us forecast the performance gain before we invested significant effort in implementation."

Intel Advisor sorted loops by potential gain, making compiler reports easier to read by showing messages on the source, and giving the project team tips for effective vectorization. It also provided key data like trip counts, data dependencies, and memory access patterns make vectorization safe and efficient.

Intel Trace Analyzer and Collector was another help in optimizing the code. This graphical tool helped the team understand MPI application behavior, quickly find bottlenecks, improve correctness—and, ultimately, maximize the tool's performance on Intel® architecture. It includes MPI communications profiling and analysis features that helped to improve weak and strong scaling.

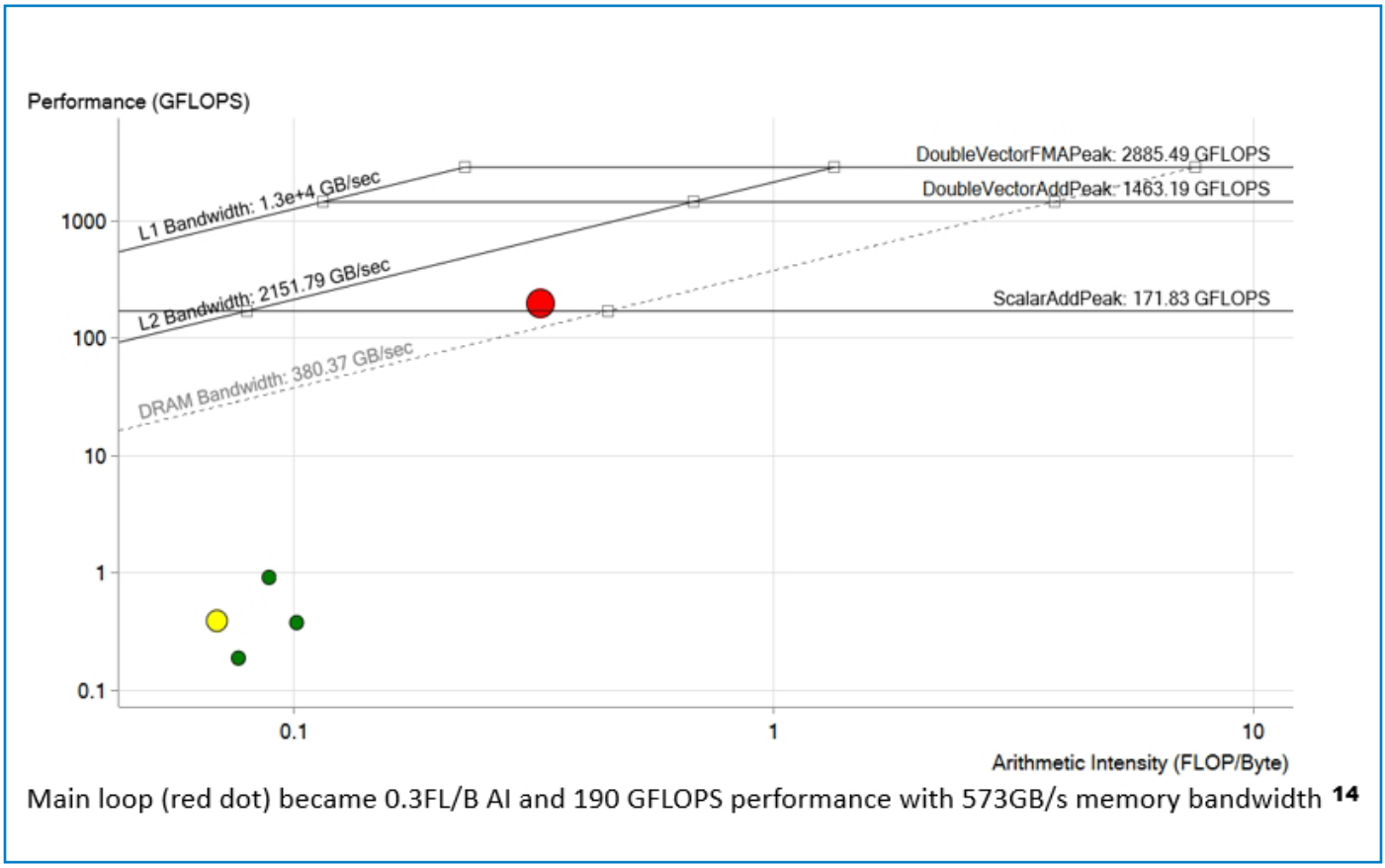

Figure 3. After optimization. *During optimization of the red loop, vectorization dependencies were removed, memory load operations optimized, and vectors and array sizes adapted for Intel Xeon Phi and AVX-512 instructions. Performance increased to 190 GFLOPS, about a 200X improvement. It is now above the DRAM limit and the scalar add limit. It is likely limited by its use of L2 cache since that is now the next roof above the loop.*

Results

Figure 4 shows a sample of the code after optimization.

After all the improvements and optimizations, the team achieved 190 GFLOPS performance and 0.3 FLOP/byte arithmetic intensity, with 100 percent mask utilization and 573 GB/s memory bandwidth.

"Using Intel Advisor and Intel Trace Analyzer and Collector, we were able to remove vector dependencies, optimize load operations, and adapt vector and array size for the Intel Xeon Phi architecture," explained Kulikov. "This optimization gave the opportunity to run 3X more variants of astrophysical tests."

Learn More

- Intel Parallel Studio XE — Create Faster Code...Faster
- Intel Advisor — Vectorization Optimization and Thread Prototyping
- Intel Trace Analyzer and Collector — MPI Tuning and Analysis

```
FYP = _mm512_mul_pd(vecincs,
                _mm512_add_pd(_mm512_sub_pd(_mm512_mul_pd(vecsr,vecfm),_mm512_mul_pd(vecsl,vecfp)),

_mm512_mul_pd(_mm512_sub_pd(vecup,vecum),_mm512_mul_pd(vecsl,vecsr))));
// down interface
vecsl = _mm512_set1_pd(Sound[index(i,k-1,l,3)]);
vecsr = _mm512_set1_pd(Sound[index(i,k,l,2)]);
vecincs = _mm512_set1_pd(1.0/(Sound[index(i,k,l,2)]-Sound[index(i,k-1,l,3)]));
vecfp = _mm512_load_pd(FY+index(i,k,l,0));
vecfm = _mm512_load_pd(FY+index(i,k-1,l,0));
vecup = _mm512_load_pd(U+index(i,k,l,0));
vecum = _mm512_load_pd(U+index(i,k-1,l,0));
FYM = _mm512_mul_pd(vecincs,
                _mm512_add_pd(_mm512_sub_pd(_mm512_mul_pd(vecsr,vecfm),_mm512_mul_pd(vecsl,vecfp)),

_mm512_mul_pd(_mm512_sub_pd(vecup,vecum),_mm512_mul_pd(vecsl,vecsr))));
```

Figure 4. Optimized code

Software