The Efficiency Optimization Study of a Geophysical Code on Manycore Computing Architectures

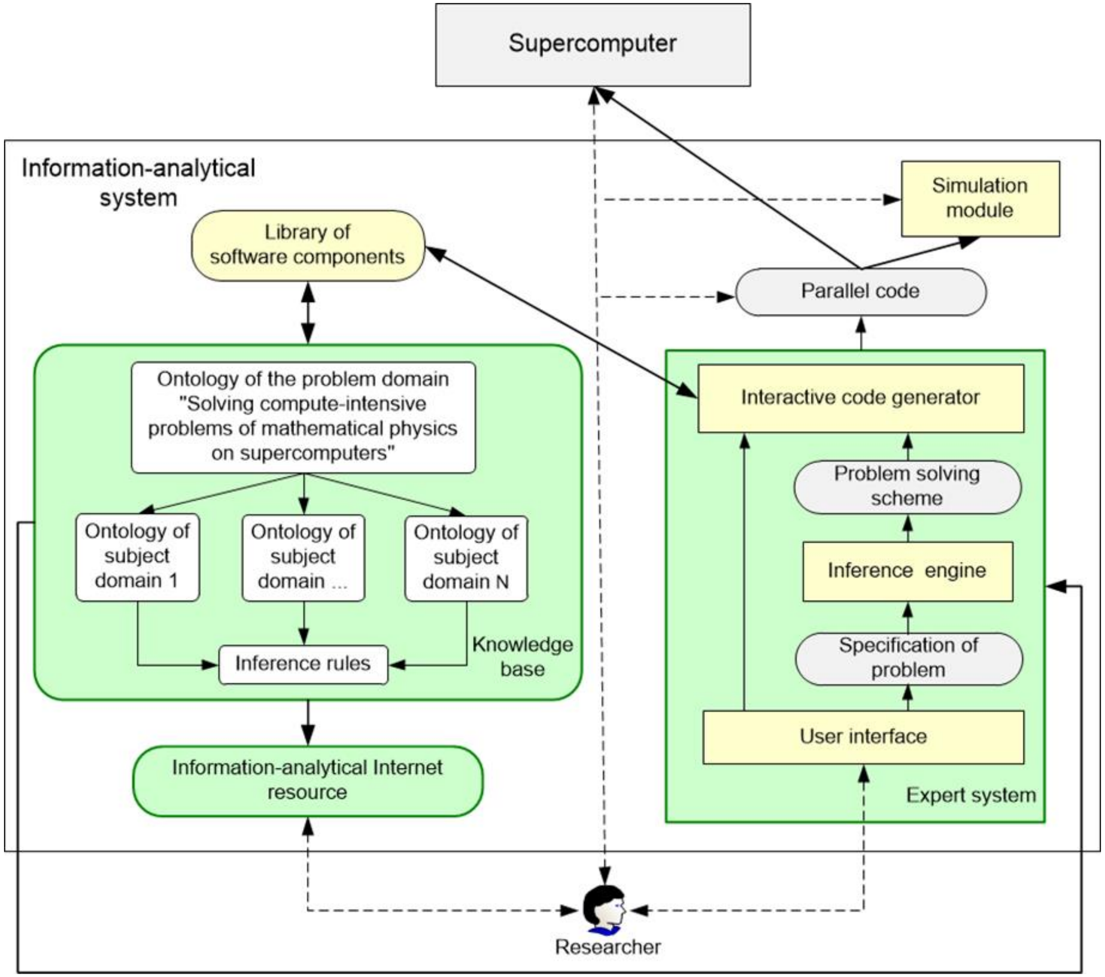
Anna Sapetina, Boris Glinskiy

Institute of Computational Mathematics and Mathematical Geophysics SB RAS, Novosibirsk

Russian Supercomputing Days (RSD) 2023

The intelligent support system for solving compute-intensive problem

- The development of efficiently parallel codes for modern supercomputer systems requires:
  - the knowledge about **relevant computational methods** and **parallel architectures** and **technologies**
  - choosing optimal numerical methods and suitable target architectures
- It is proposed to present the accumulated knowledge about solving compute-intensive problems in an **ontological form** [1]
- **Similar projects:** AlgoWiki, LuNA, information resources based on ontological descriptions



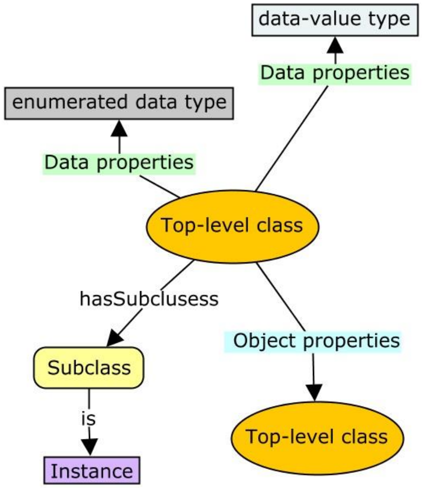
Supercomputer
Information-analytical system
Library of software components
Simulation module
Parallel code
Ontology of the problem domain "Solving compute-intensive problems of mathematical physics on supercomputers"
Ontology of subject domain 1
Ontology of subject domain ...
Ontology of subject domain N
Inference rules
Knowledge base
Interactive code generator
Problem solving scheme
Inference engine
Specification of problem
User interface
Expert system
Information-analytical Internet resource
Researcher

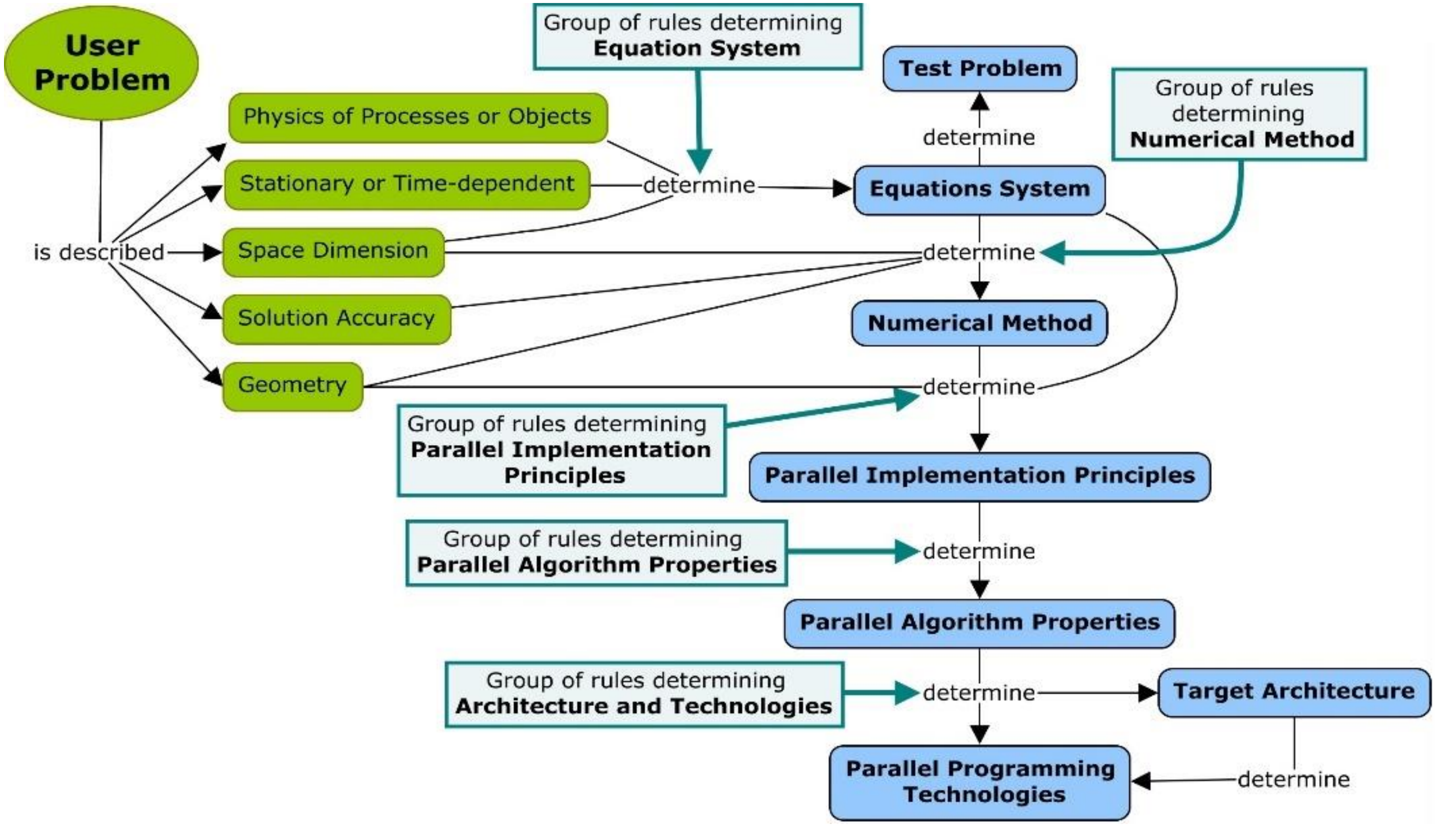
Ontology is a formal explicit description of the domain terms and the relationship between them.

[1] B. Glinskiy, Y. Zagorulko, G. Zagorulko, I. Kulikov, A. Sapetina. The Creation of Intelligent Support Methods for Solving Mathematical Physics Problems on Supercomputers. CCIS, vol. 1129, pp. 427–438, 2019

The intelligent support system realization

- Developed **top-level ontology** for solving compute-intensive problems of astrophysics, geophysics and plasma physics
- The ontology of mathematical methods and parallel algorithms and the ontology of parallel architectures and technologies, **together with the inference rules formulated by experts in a given subject area**, make it possible to select an efficient numerical method, a parallel algorithm, and a computational architecture for solving a user's problem.




User Problem
is described
Physics of Processes or Objects
Stationary or Time-dependent
Space Dimension
Solution Accuracy
Geometry
Group of rules determining Equation System
determine
Test Problem
determine
Equations System
Group of rules determining Numerical Method
determine
Numerical Method
determine
Group of rules determining Parallel Implementation Principles
Parallel Implementation Principles
Group of rules determining Parallel Algorithm Properties
determine
Parallel Algorithm Properties
Group of rules determining Architecture and Technologies
determine
Target Architecture
Parallel Programming Technologies
determine

The propagation of seismic waves in complicated elastic inhomogeneous media

$\vec{U} = (U, V, W)^T$− displacements vector

$$\rho \frac{\partial^2 \vec{U}}{\partial t^2} = [C]\vec{U} + \vec{F}(t, x, y, z)$$

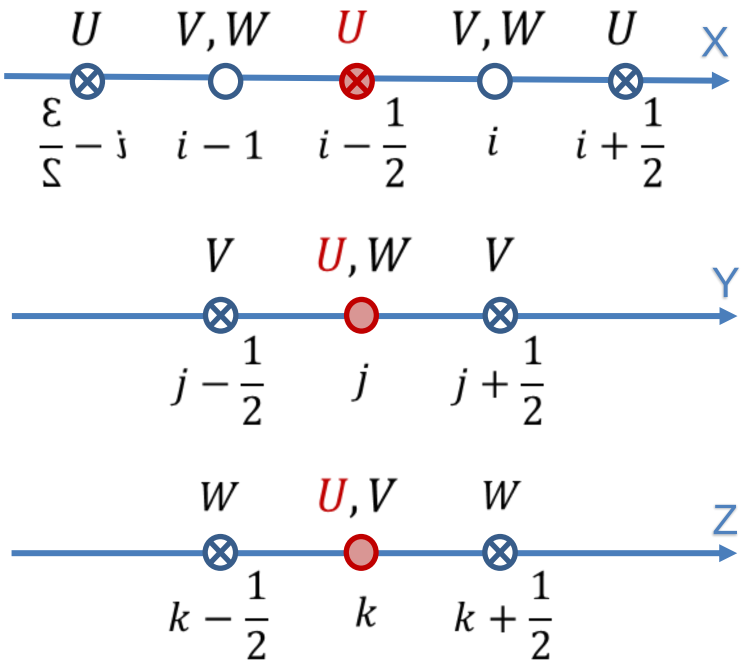
$$C = \begin{bmatrix} (\lambda + 2\mu)\frac{\partial^2}{\partial x^2} + \mu\left(\frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}\right) & (\lambda + \mu)\frac{\partial^2}{\partial x \partial y} & (\lambda + \mu)\frac{\partial^2}{\partial x \partial z} \\ (\lambda + \mu)\frac{\partial^2}{\partial y \partial x} & (\lambda + 2\mu)\frac{\partial^2}{\partial y^2} + \mu\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial z^2}\right) & (\lambda + \mu)\frac{\partial^2}{\partial y \partial z} \\ (\lambda + \mu)\frac{\partial^2}{\partial z \partial x} & (\lambda + \mu)\frac{\partial^2}{\partial z \partial y} & (\lambda + 2\mu)\frac{\partial^2}{\partial z^2} + \mu\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) \end{bmatrix}$$

$\vec{U}\big|_{t=0}$=0, $\sigma_{xz}|_{z=0}$=0, $\sigma_{yz}\big|_{z=0}$=0, $\sigma_{zz}|_{z=0}$=0

where t is the time, ρ(x, y, z) is the density, λ(x, y, z), $\mu(x, y, z)$ are the Lame coefficients

Numerical solution of elastodynamic equations

- Explicit finite difference scheme on staggered grids
- 2nd order of approximation with respect to time and space
- Memory bound problem: the limiting factor is a speed of access to memory



Size of grids for test calculations:

- 581×581×581 (~11 Gb)
- 581×581×193 (~3,6 Gb)

The approach to the construction of the scheme is described in an article: Bihn M., Weiland T. A. Stable Discretization Scheme for the Simulation of Elastic Waves // Proceedings of the 15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics (IMACS 1997). T. 2, C. 75-80.

Considered Manycore Systems:

Intel and IBM Processors

Intel Broadwell[1]	Processor	2 × Intel Xeon E5-2697A v4 (2.6 GHz, 16 cores, SMT2)
	Memory	128 GB DDR4 RAM
	Peak performance	1 331 GFLOPS
Intel KNL[1]	Processor	Intel Xeon Phi 7290 KNL (1.5 GHz, 72 cores, SMT4)
	Memory	16 GB MCDRAM, 96 GB DDR4 RAM
	Peak performance	3 456 GFLOPS
IBM POWER9	Processor	IBM POWER9 Proc. (3.8 GHz, 2×12 core Typical, SMT8)
	Memory	32×32 GB DDR4 RAM
	Peak performance	2 918 GFLOPS

Key Features:

- a small number of cores (several decades)
- high clock frequency of cores
- vector process units
- support simultaneous multithreading (SMT)

[1] This systems is a part of clusters NKS-1P of **SSCC ICMMG SB RAS**

Parallelization and optimizations:

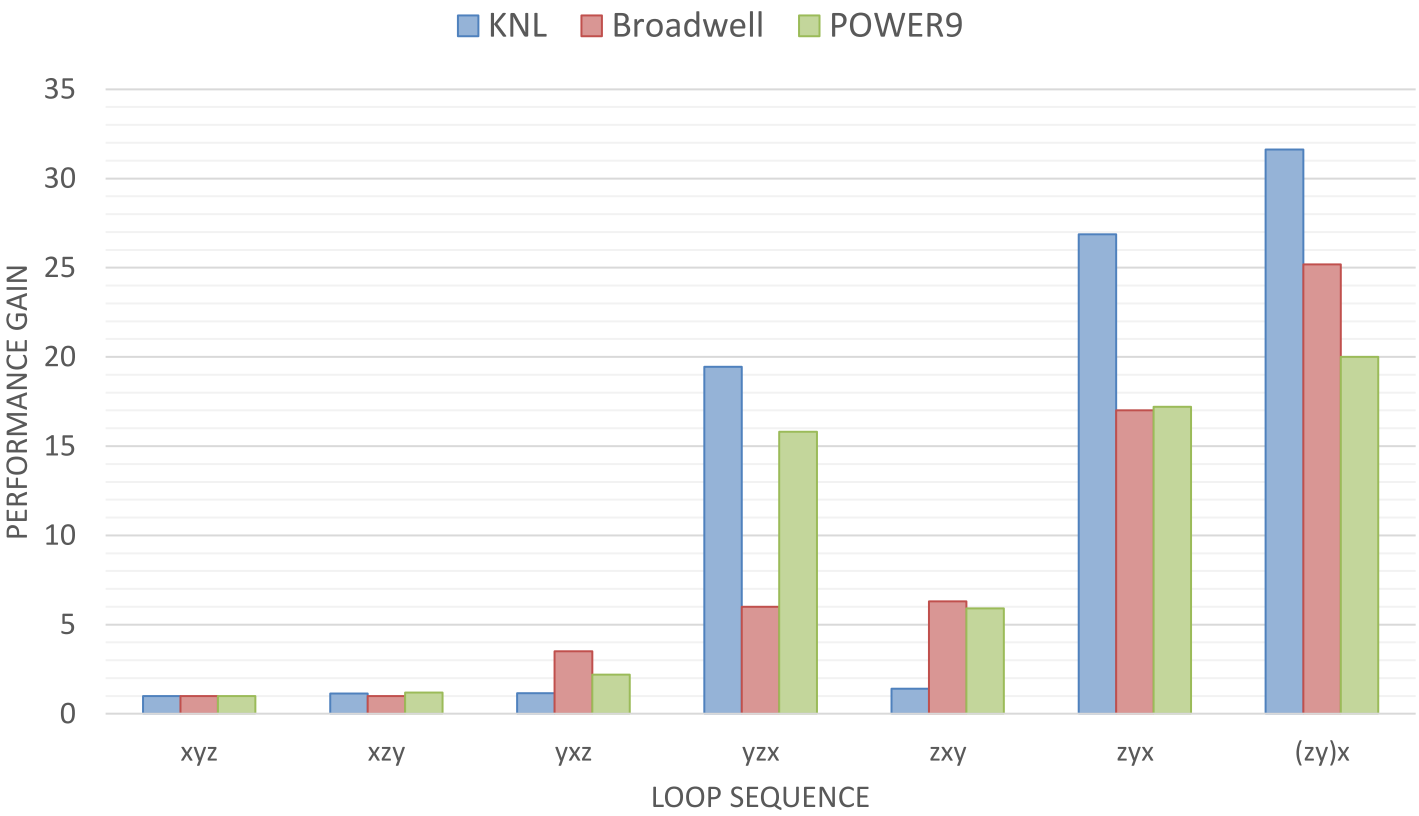
Intel and IBM Processors

- **Aligned**: posix_memalign
- **External loop** is parallelized using OpenMP
- **Internal loop** is vectorized
  - for Intel with AVX2/AVX-512 technologies
  - Auto-vectorization VS Intrinsic functions
  - Vector operations speed up the program several times (2.75 times for KNL)

```
for all time steps do
          #pragma omp paraller for…
          for all X points do
                    for all Y points do
                              #pragma simd
                              for all Z points do
                                        U,V,W computations
                              end for
                    end for
          end for
          Snapshot check
end for
```

Parallelization and optimizations:
Intel and IBM Processors

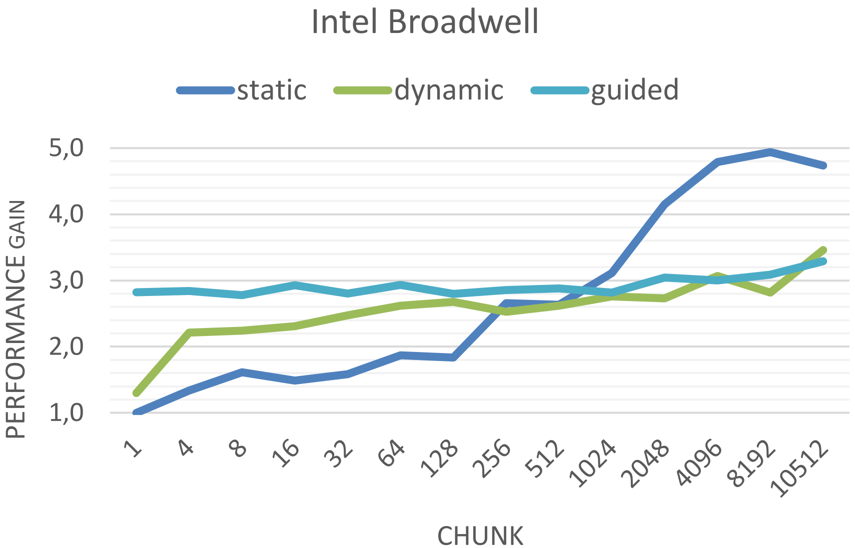
- **Caching:** different loops sequence – xyz / xzy / yxz / yzx / zxy / zyx
- **Load balancing:** collapse of two external loops in one – (zy)x

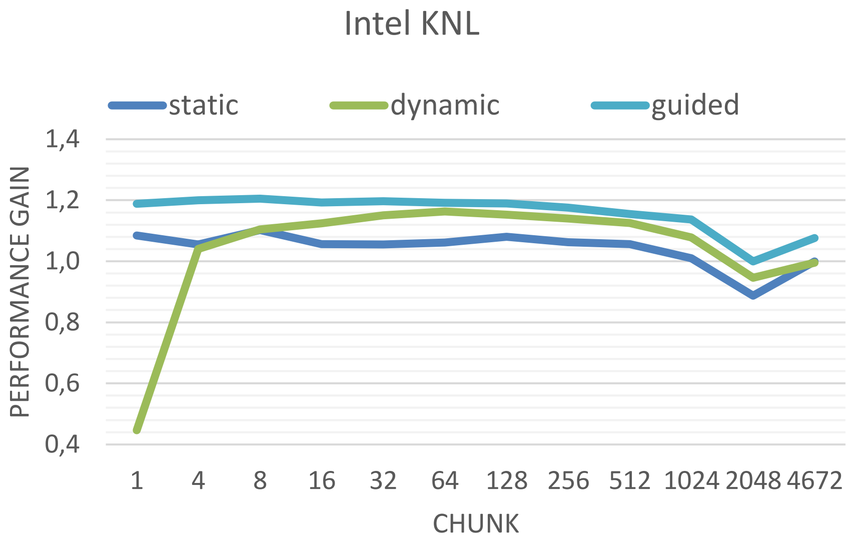


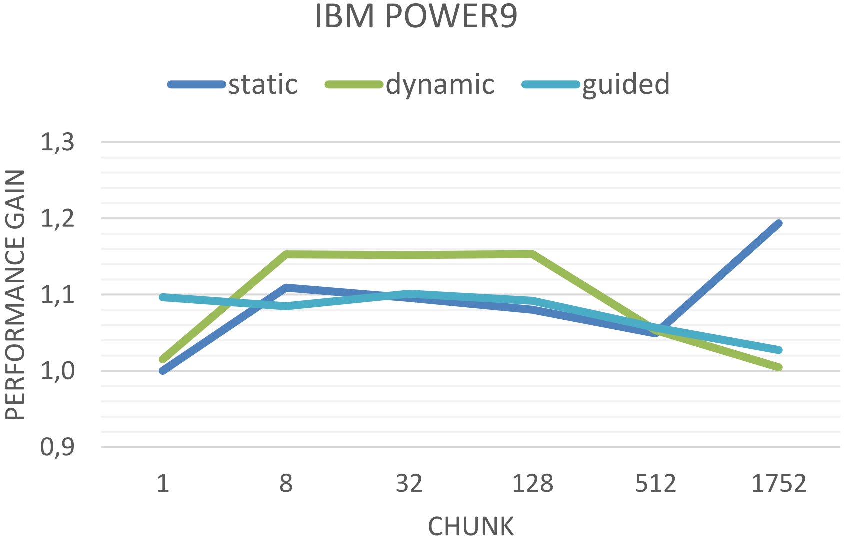
Parallelization and optimizations:

Intel and IBM Processors

- **Load balance:** choosing the OpenMP schedule type and chunk size


Intel Broadwell

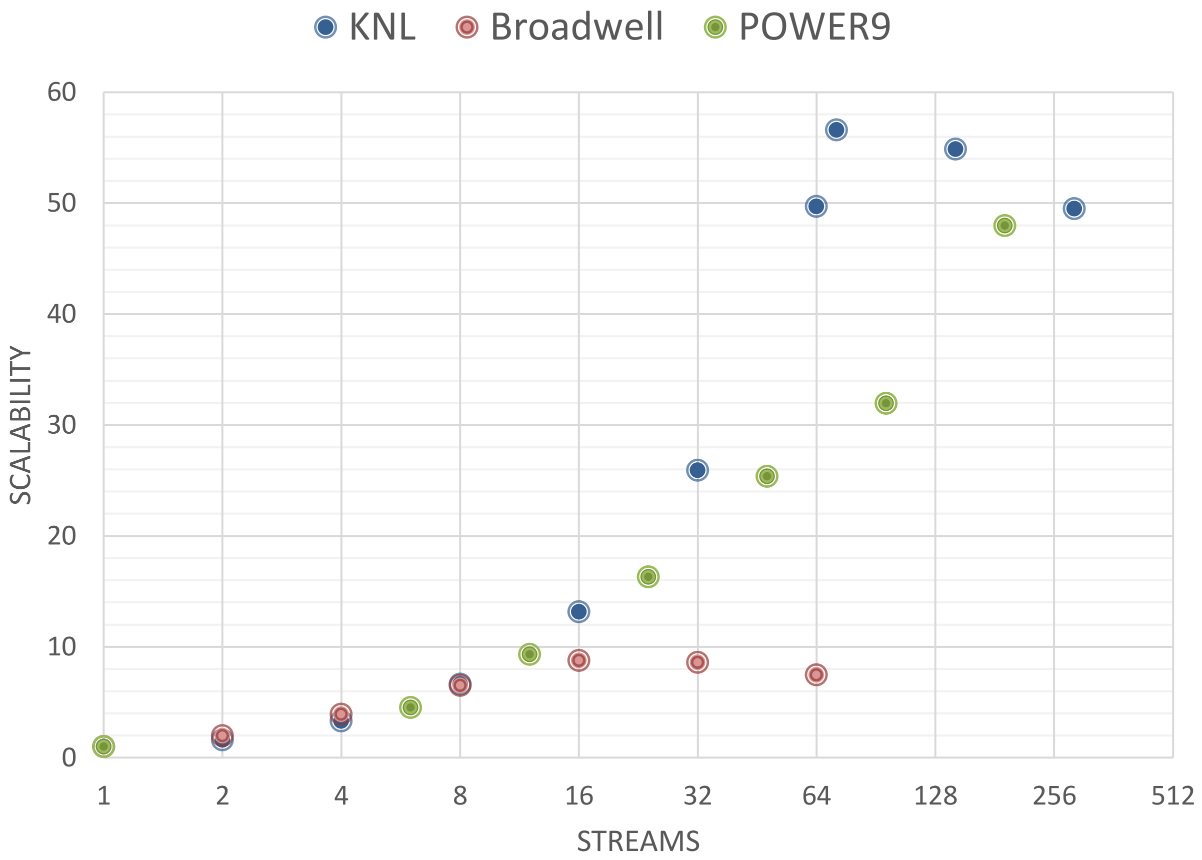

Intel KNL


IBM POWER9

Parallelization and optimizations:

Intel and IBM Processors

- Flat memory mode for Intel KNL: acceleration in 1,3 times
- Strong scalability and the use of simultaneous multithreading



Considered Manycore Systems:

NVIDIA accelerators

NVIDIA Fermi[1]	Accelerator	NVIDIA Tesla M2090 (1,3 GHz, 512 cores, 6 GB GDDR5)
	Memory	96 GB DDR4 RAM
	GPU Peak	1 331 GFLOPS
NVIDIA Kepler[1]	Accelerator	NVIDIA Tesla K40 (0,75 GHz, 2880 cores, 12 GB GDDR5)
	Memory	64 GB DDR4 RAM
	GPU Peak	4 291 GFLOPS
NVIDIA Pascal[2]	Accelerator	NVIDIA Tesla P100 (1,48 GHz, 3584 cores, 16 GB HBM2)
	Memory	256 GB DDR4 RAM
	GPU Peak	10 608 GFLOPS

Key Features:

- a large number of cores (hundreds and thousands)
- highly simplified cores operating at a low frequency
- complex memory system
- it is necessary to use threads many times larger than the number of cores

[1] This systems is a part of clusters NKS-30T+GPU of SSCC ICMMG SB RAS (Rpeak– 85 TFLOPS)

[2] This systems is a part of hybrid clusters of CC FED RAS (Rpeak – 56 TFLOPS)

Parallelization and optimizations:
NVIDIA accelerators

- All calculations perform on the GPU
- Aligned: CudaMalloc3D
- Dimension and size of a thread block (Bsize)
  - a 3D grid of blocks
  - size for the component X must be a multiple of the length of the warp
  - the specific size of a thread block are chosen empirically for each algorithm
  - it possible to speed up the work of the program several times

Parallelization and optimizations:

Tesla M2090

BSize	PGain	BSize	PGain	BSize	PGain	BSize	PGain
512		256		128		64	
8×8×8	0.52						
16×8×4	1.0	16×4×4	1.05	16×4×2	0.96	16×2×2	0.99
32×4×4	1.47	32×4×2	1.49	32×2×2	1.48	32×2×1	1.26
64×4×2	1.42	64×2×2	1.65	64×2×1	1.38	64×1×1	1.35
128×2×2	1.45	128×2×1	1.44	128×1×1	1.41		
256×2×1	1.2	256×1×1	1.41				
512×1×1	0.98						

Tesla K40

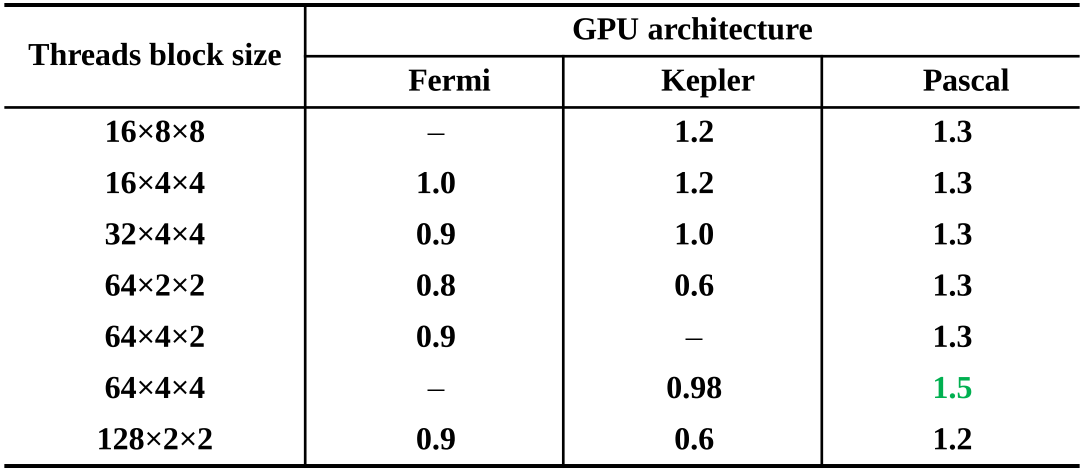
BSize	PGain	BSize	PGain	BSize	PGain	BSize	PGain
1024		512		256		128	
16×8×8	1.0	16×8×4	1.11	16×4×4	1.04	16×4×2	1.02
32×8×4	1.53	32×4×4	1.67	32×4×2	1.63	32×2×2	1.55
64×4×4	1.64	64×4×2	1.7	64×2×2	1.82	64×2×1	1.72
128×4×2	1.63	128×2×2	1.7	128×2×1	1.78	128×1×1	1.73
256×2×2	1.44	256×2×1	1.58	256×1×1	1.69		
512×2×1	1.13	512×1×1	1.43				

Tesla P100

BSize	PGain	BSize	PGain	BSize	PGain	BSize	PGain
1024		512		256		128	
16×8×8	1.0	16×8×4	1.02	16×4×4	1.02	16×4×2	0.98
32×8×4	1.25	32×4×4	1.29	32×4×2	1.23	32×2×2	1.23
4×64×4	0.49	32×8×2	1.23				
4×4×64	0.48	32×16×1	1.12				
64×4×4	1.24	64×4×2	1.26	64×2×2	1.25	64×2×1	1.08
128×4×2	1.25	128×2×2	1.27	128×2×1	1.12	128×1×1	1.08
256×2×2	1.22	256×2×1	1.14	256×1×1	1.13		
512×2×1	1.1	512×1×1	1.14				

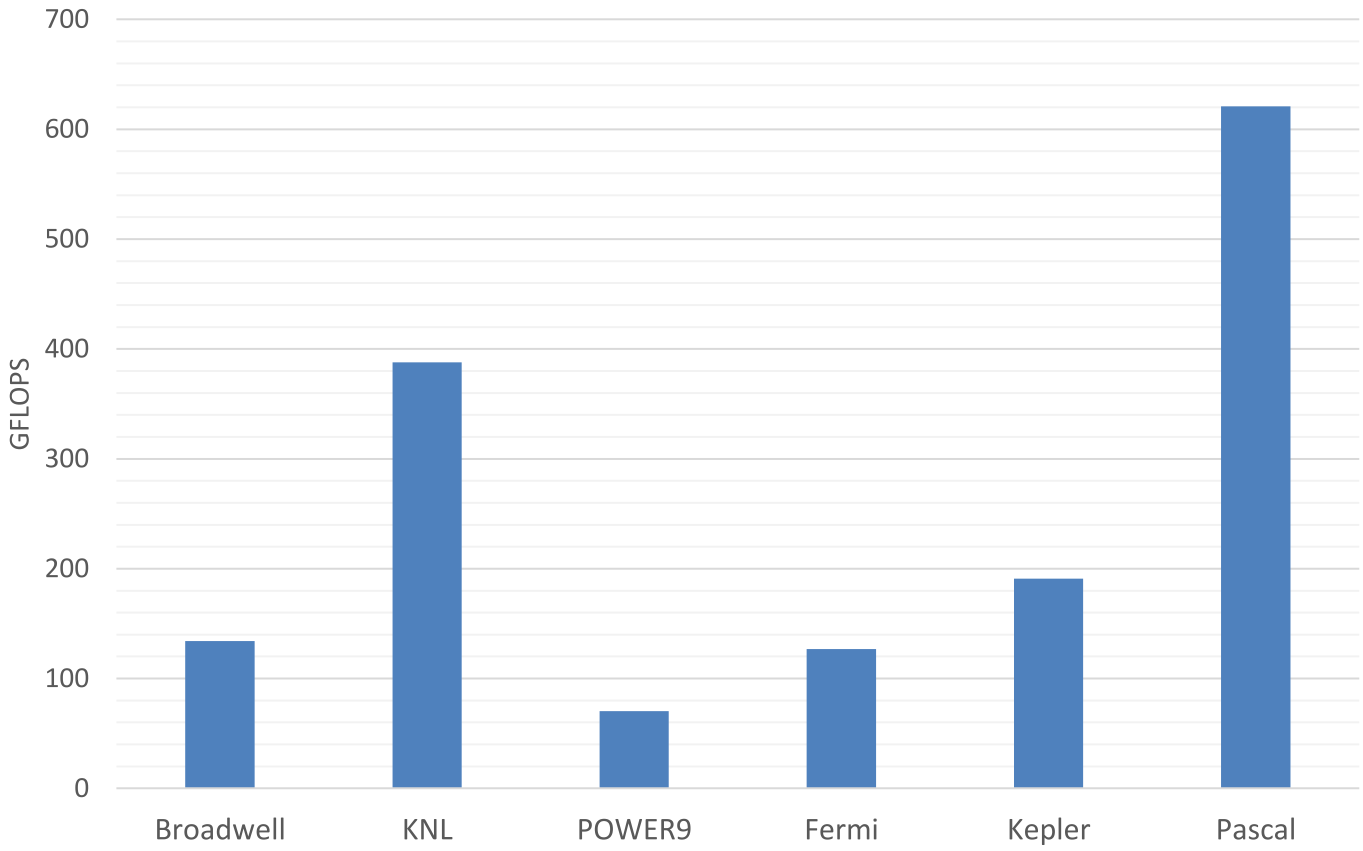
Parallelization and optimizations:
NVIDIA accelerators

- Constant memory: save the main constants used at each time step (Pgain about 4 %)
- Shared memory
    - No effect on Fermi and Kepler
    - Acceleration in 1,48 times on P100



Threads block size	GPU architecture		
	Fermi	Kepler	Pascal
16×8×8	–	1.2	1.3
16×4×4	1.0	1.2	1.3
32×4×4	0.9	1.0	1.3
64×2×2	0.8	0.6	1.3
64×4×2	0.9	–	1.3
64×4×4	–	0.98	1.5
128×2×2	0.9	0.6	1.2

Performance Comparison



Rpeak: 1331 3456 2918 1331 4291 10608 GFLOPS

Expert rules for choosing CPU optimizations

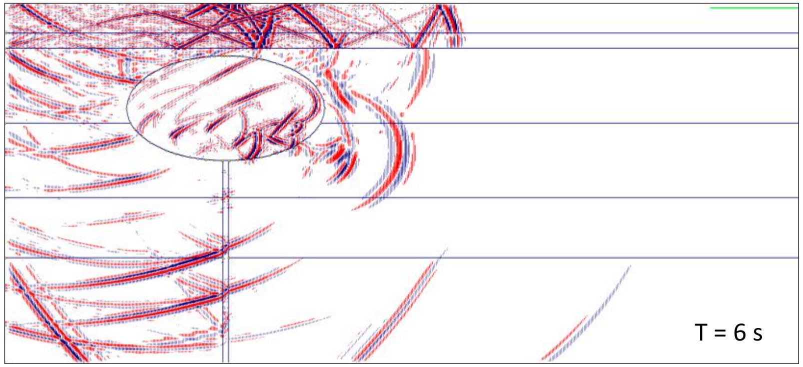
1. For 2D and 3D problem codes the inner loop of the algorithm should be vectorized, and the outer loop in space should be parallelized using OpenMP.
2. For memory access efficiency, all central arrays must be aligned in memory, for example, using the `posix_memalign` or `_mm_malloc` functions.
3. It is necessary to mark up the code using vectorization directives or use intrinsic for efficient vectorization to fit the processor's vector registers size.
4. For better caching and load balancing in 3D codes, it is best to choose a nested sequence of (zy)x loops, where the z and y loops are combined into one.
5. When using OpenMP to parallelize finite difference codes for multicore systems, it is better to use schedule static with a maximum chunk size (default), for manycore systems – schedule guided with a short chunk size.
6. When using SMT for parallelizing finite difference codes, 1 thread per core is better for Intel processors, and the max thread number per core is better for IBM processors.
7. When using KNL to execute finite difference codes, it is preferable to use flat memory mode instead of cache memory mode with all the main arrays placed in MCDRAM memory.
8. If possible, the choice of manycore computing accelerators is preferable.

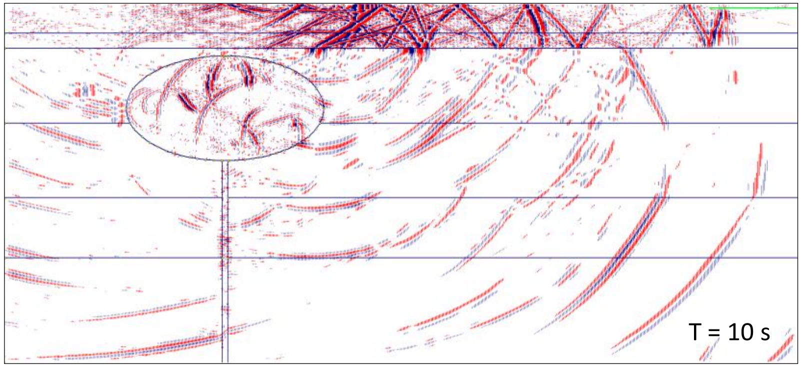
Expert rules for choosing GPU optimizations

1. If possible, it is better to load the entire problem into the memory of the GPUs
2. The dimension of the grid of blocks must match the dimension of the problem
3. The block size for 3D finite difference problems is better to use equal to 32 or 64 for the x component and equal to 4 or 2 for the y and z components. The maximum possible block size is not necessarily better.
4. Use constant memory to store frequently reused constants. The more of them, the more this type of storage is preferable.
5. The use of shared memory for a small data reuse (memory-bound problem) is justified on newer GPUs starting from the Pascal architecture.
6. If possible, the choice of multi-core computing accelerators is preferable

Conclusion

- The main nuances in the development of high-performance software for clusters with various manycore processors and accelerators have been investigated using the example of solving the geophysical problem of elastic wave propagation in the three-dimensional elastic media.
- The effect on the performance of different code optimizations is investigated.
- A software code has been developed, with a performance of about 390 GFLOPS for Intel KNL and a software code with a performance of about 620 GFLOPS for the NVIDIA Tesla P100.
- The expert rules are formulated for choosing development optimizations for various manycore architectures in the intelligent support system for solving compute-intensive problems of mathematical physics.





Thanks for your attention!