Some perspectives on high-performance computing in the Geosciences

Xing Cai

Simula Research Laboratory & Univ. Oslo

Geilo, January 19, 2012

The main question

What's the achievable performance of a scientific code on modern parallel hardware (multicore CPUs and GPUs)?

A related question: How to effectively use modern parallel H/W?

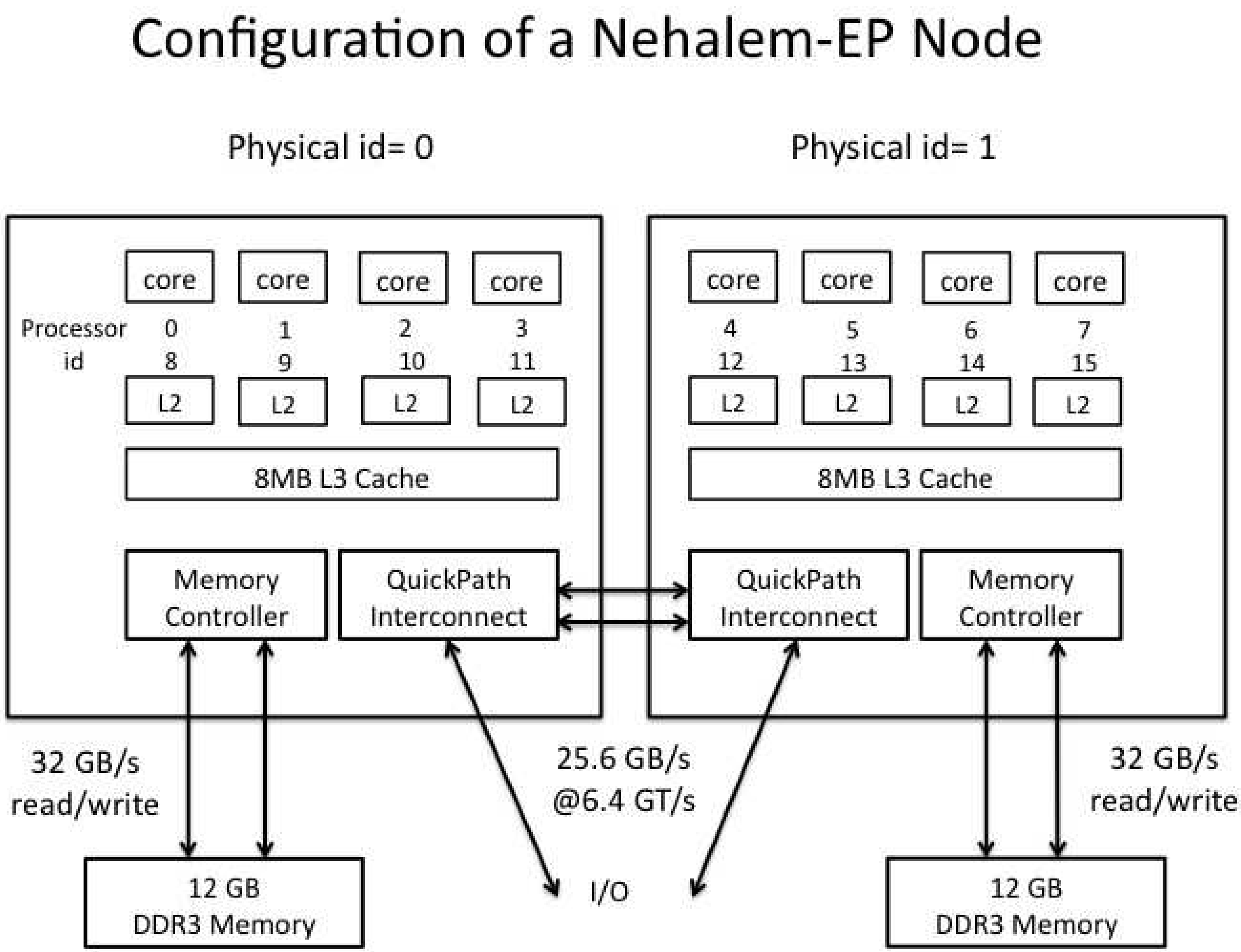
Motivation

- GFLOPS—giga 10^9 floating-point operations achieved per second—the most widely used metric for code performance
- Quite often, the achieved GFLOPS rate is far below the theoretical peak
  - Is this supposed to be what we should achieve?
- This presentation
  - gives a simple performance analysis/prediction strategy, and
  - reports its application in the context of basin-filling simulations

Performance prediction on multicore CPUs

- When a scientific code is executed on a computer:
  - Floating-point (FP) operations are carried out on values provided through a data path consisting of several links
  - Different amounts of data pass through different links
  - FPs and data transfers can happen at the same time, thanks to pipelining and data prefetching
- What is the performance limiting factor?
  - The CPU's floating-point capability?
  - Memory bandwidth?
  - Read/write bandwidth between registers and L1 cache?
  - Something else?
- The answer depends, of course!
- We want a **simple analysis** that can identify the bottleneck, and in addition, **roughly predict** the computing time on a given multicore CPU

Nehalem-EP: an example of multicore architecture



Can we predict the computing time?

The answer is yes...

- if we know for software,
  - n_{FP} — # floating-point operations
  - n_{LD} — # loads from L1 cache to registers
  - n_{ST} — # stores from registers to L1 cache
  - n^{M}_{2way} — # reads+ writes between memory and last-level cache
- if we know for hardware,
  - F — peak floating-point capability
  - B^{r}_{L1} — load bandwidth from L1 cache to registers
  - B^{w}_{L1} — store bandwidth from registers to L1 cache
  - B_M — 2-way bandwidth of main memory

Simplified prediction models

The case of using a single core:

$$\max\left(\frac{n_{\rm FP}}{F},\frac{n_{\rm LD}}{B^r_{L1}},\frac{n_{\rm ST}}{B^w_{L1}},\frac{n^M_{\rm 2way}}{B_M}\right).$$

The case of using p cores:

$$\max\left(\frac{n_{\rm FP}}{pF},\frac{n_{\rm LD}}{pB^r_{L1}},\frac{n_{\rm ST}}{pB^w_{L1}},\frac{n^M_{\rm 2way}}{B^p_M}\right).$$

How to use?

- Need to know the computation and memory complexity
  - The numerical algorithm itself provides approximate counts of n_{FP} and n_{LD}
  - Hardware performance counters (e.g. via the PAPI tool) give precise counts
  - Estimation needed for n^{M}_{2way}
- Need to know the hardware characteristics
  - Hardware specifications (FP & L1 cache)
  - Standard simple benchmark of memory bandwidth

Advantages and weaknesses

- Advantages
  - simple philosophy — a quick characteristic overview
  - capable of identifying the (switching) performance bottleneck
  - no need for detailed analysis of cache misses
  - can even predict the performance before actual code implementation
    - easy to find n_{FP}, n_{LD}, n_{ST} (and n^{M}_{2way}), which are independent of problem size
    - F, B^{r}_{L1}, B^{w}_{L1}, B_M are readily known (through hardware spec. and STREAM benchmarking)
- Weaknesses
  - Very crude predictions — lower bound of time usage
  - No consideration of stall of cycles due to unavailable H/W resource
  - No consideration of the actual parallelization strategy (MPI, OpenMP, Pthreads) and communication/synchronization overhead

An example of solving 3D heat equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} + f$$

Fully-explicit numerical scheme ($\Delta x = \Delta y = \Delta z = h$):

$$\frac{u^{\ell+1}_{i,j,k} - u^{\ell}_{i,j,k}}{\Delta t}$$

$$= \frac{u^{\ell}_{i-1,j,k} + u^{\ell}_{i,j-1,k} + u^{\ell}_{i,j,k-1} - 6u^{\ell}_{i,j,k} + u^{\ell}_{i+1,j,k} + u^{\ell}_{i,j+1,k} + u^{\ell}_{i,j,k+1}}{h^2} + f_{i,j,k}$$

The C code

```
   t = 0.;
   while (t<T) {
#pragma omp for private(i,j) schedule(static)
     for (k=1; k<n-1; k++)
       for (j=1; j<n-1; j++)
         for (i=1; i<n-1; i++)
           u_new[k][j][i] = u_old[k][j][i] + rhs[k][j][i]
             + factor*(u_old[k][j][i-1]+u_old[k][j][i+1]
                      +u_old[k][j-1][i]+u_old[k][j+1][i]
                      +u_old[k-1][j][i]+u_old[k+1][j][i]
                      -6*u_old[k][j][i]);

#pragma omp single
     {
       /* pointer swap */
       /* ... */
       t += dt;
     }
   }
```

Predicting performance

Testbed: a compute node consisting of two quad-core Intel Xeon 2GHz E5504 CPUs

- $F = 4$ GFLOPS (for a non-SIMD compiler), $B^r_{L1} = B^w_{L1} = 16$ GB/s
- Per time step, per grid point: $n_{\rm FP} = 10$, $n_{\rm LD} = 11 \times 8$ bytes, $n_{\rm ST} = 0$, $n^M_{2way} = 3 \times 8$ bytes
- B^p_M values are measured by STREAM

# cores	1	2	4	6	8
B^p_M	6.22 GB/s	12.19 GB/s	13.89 GB/s	13.24 GB/s	13.03 GB/s
Mesh size: $99 \times 99 \times 99$, # time steps: 60001					
T_A	358.32 s	184.84 s	120.72 s	114.61 s	122.14 s
T_P	320.20 s	160.10 s	100.59 s	105.53 s	107.23 s

T_A: actual time usage, T_P: predicted time usage

Observations

For the fully-explicit finite difference 3D heat solver:

- Floating-point operations are never the performance bottleneck
- Data transfer is indeed the bottleneck
- However, the main memory is not always the bottleneck
- For a small number of cores in use, the main memory bandwidth is sufficient, in comparison with the aggregate bandwidth between L1 and registers
- For a large number of cores in use, the main memory bandwidth becomes the bottleneck

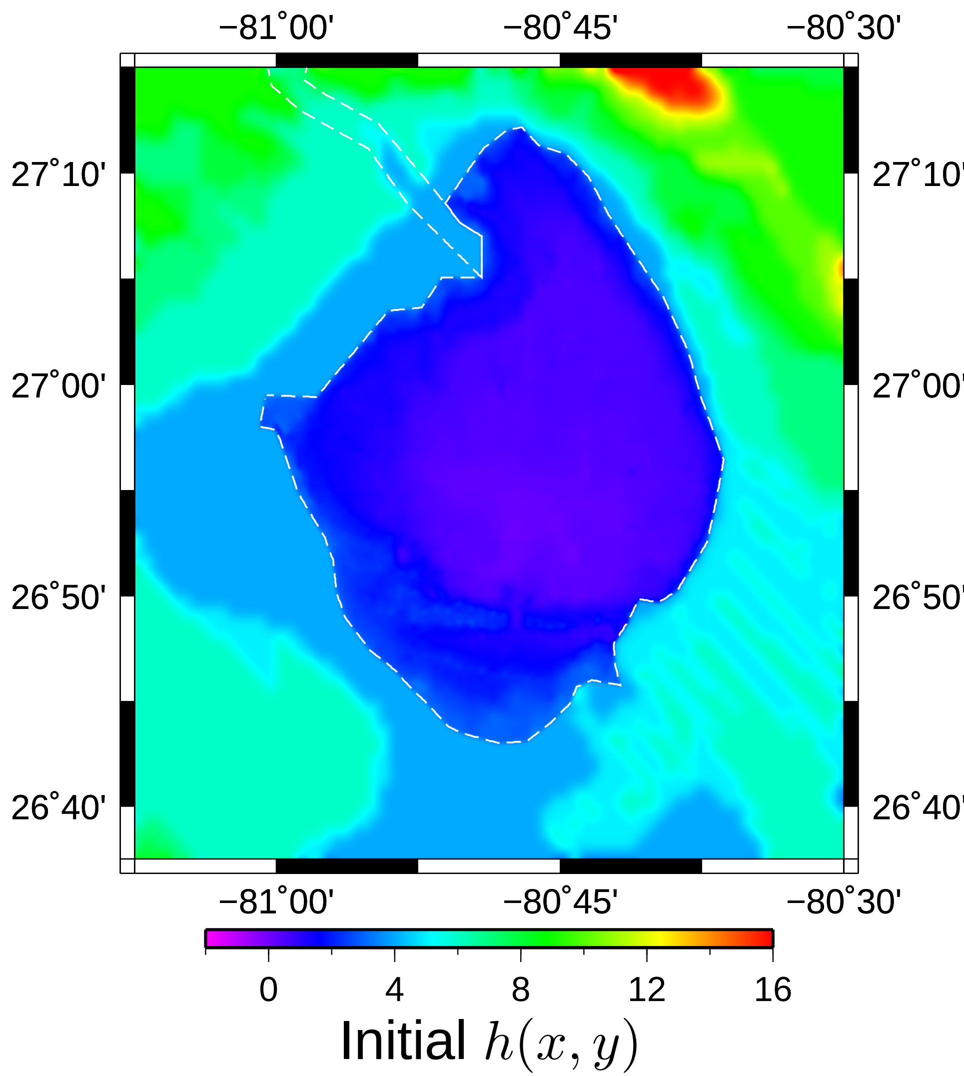
Math model of sediment transport

$$\frac{\partial h}{\partial t} = \frac{1}{C_s}\nabla \cdot (\alpha s \nabla h) + \frac{1}{C_m}\nabla \cdot (\beta(1-s)\nabla h), \quad (1)$$

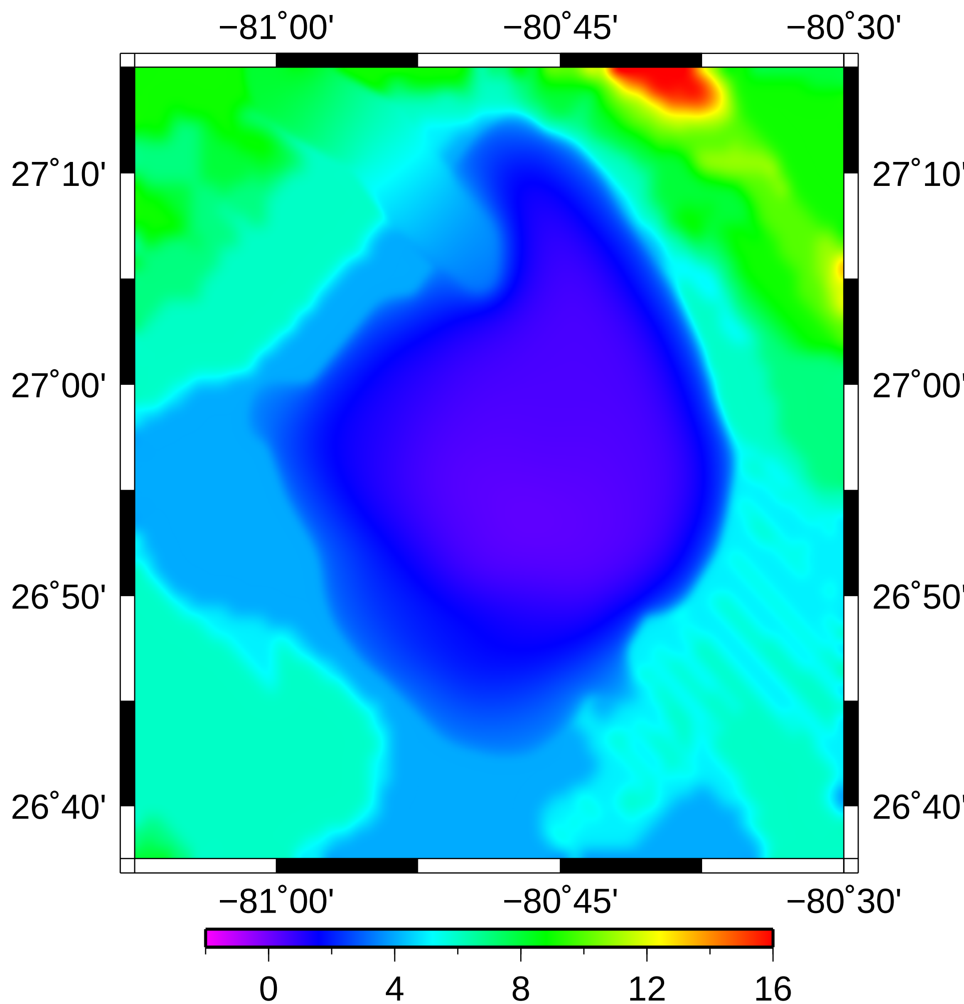
$$A\frac{\partial s}{\partial t} + s\frac{\partial h}{\partial t} = \frac{1}{C_s}\nabla \cdot (\alpha s \nabla h). \quad (2)$$

- Two lithologies (sand and mud) considered in sedimentary basin filling
- $h(x, y, t)$ — height
- $s(x, y, t)$ — fraction of sand

Example of Lake Okeechobee, Florida

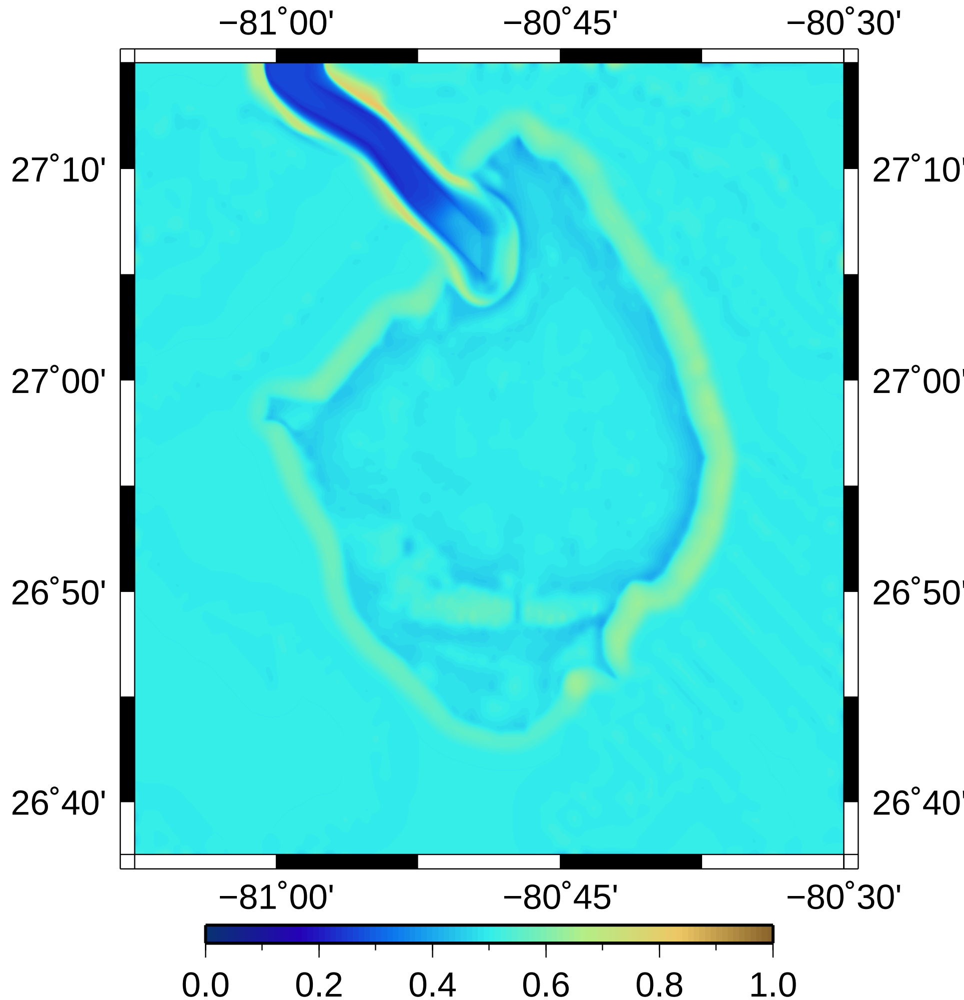


Example of Lake Okeechobee, Florida (cont'd)



Solution of $h(x, y)$ after 1,000 years

Example of Lake Okeechobee, Florida (cont'd)



Solution of $s(x, y)$ after 1,000 years

A fully-explicit scheme

$$\frac{h^{\ell+1} - h^{\ell}}{\Delta t} = \frac{1}{C_s}\nabla \cdot (\alpha s^{\ell} \nabla h^{\ell}) + \frac{1}{C_m}\nabla \cdot (\beta(1 - s^{\ell})\nabla h^{\ell}),$$

$$A\frac{s^{\ell+1} - s^{\ell}}{\Delta t} + s^{\ell+1}\frac{h^{\ell+1} - h^{\ell}}{\Delta t} = \frac{1}{C_s}\nabla \cdot (\alpha s^{\ell} \nabla h^{\ell+1}).$$

- Straightforward calculations
- No need to solve linear systems
- Inferior numerical stability

Semi-implicit scheme 1

$$\frac{h^{\ell+1}-h^{\ell}}{\Delta t} = \frac{1}{C_s}\nabla\cdot(\alpha s^{\ell}\nabla h^{\ell+1}) + \frac{1}{C_m}\nabla\cdot(\beta(1-s^{\ell})\nabla h^{\ell+1}),$$

$$A\frac{s^{\ell+1}-s^{\ell}}{\Delta t} + s^{\ell+1}\frac{h^{\ell+1}-h^{\ell}}{\Delta t} = \frac{1}{C_s}\nabla\cdot(\alpha s^{\ell+1}\nabla h^{\ell+1}).$$

- $h^{\ell+1}$ and $s^{\ell+1}$ are updated separately
- One linear system wrt $h^{\ell+1}$
- One linear system wrt $s^{\ell+1}$

Semi-implicit scheme 2

$$\frac{h^{\ell+1,k}-h^{\ell}}{\Delta t} = \frac{1}{2}\left(\frac{1}{C_s}\nabla\cdot(\alpha s^{\ell+1,k-1}\nabla h^{\ell+1,k}) + \frac{1}{C_m}\nabla\cdot(\beta(1-s^{\ell+1,k-1})\nabla h^{\ell+1,k})\right) + \frac{1}{2}\left(\frac{1}{C_s}\nabla\cdot(\alpha s^{\ell}\nabla h^{\ell}) + \frac{1}{C_m}\nabla\cdot(\beta(1-s^{\ell})\nabla h^{\ell})\right)$$

$$A\frac{s^{\ell+1,k}-s^{\ell}}{\Delta t} + \frac{s^{\ell+1,k}+s^{\ell}}{2}\frac{h^{\ell+1,k}-h^{\ell}}{\Delta t} = \frac{1}{2}\left(\frac{1}{C_s}\nabla\cdot(\alpha s^{\ell+1,k}\nabla h^{\ell+1,k}) + \frac{1}{C_s}\nabla\cdot(\alpha s^{\ell}\nabla h^{\ell})\right)$$

- Crank-Nicolson strategy is used
- Inner iterations, $k = 1, 2, \ldots$, are used within each time step
- Numerical experiments show that $k = 2$ can give second-order accuracy in time

Fully-implicit scheme 1

- $h^{\ell+1}$ and $s^{\ell+1}$ are updated simultaneously
- Backward Euler temporal discretization is used
- A nonlinear system arises per time step

$$\begin{aligned} \mathbf{F}_h\left(\mathbf{h}^{\ell+1}, \mathbf{s}^{\ell+1}, \mathbf{h}^{\ell}, \mathbf{s}^{\ell}\right) &= \mathbf{0}, \\ \mathbf{F}_s\left(\mathbf{h}^{\ell+1}, \mathbf{s}^{\ell+1}, \mathbf{h}^{\ell}, \mathbf{s}^{\ell}\right) &= \mathbf{0}. \end{aligned}$$

- Newton iterations are needed

Fully-implicit scheme 2

- Same as fully-implicit scheme 1, except that Crank-Nicolson temporal discretization is used
- Second-order accuracy in time

Comparison of the five schemes

Scheme	Action	n_{FP}	n_{LD}	n_{ST}	n^{M}_{2way}
Fully	Compute $h_{i,j}$	57	43	1	5
-explicit	Compute $s_{i,j}$	37	24	1	5
Semi-	Set up h system	62	21	8	10
implicit 1	Solve h system	15	52	9	21
	Set up s system	35	35	10	10
	Solve s system	15	68	14	21
Semi-	Set up h system	262	158	24	20
implicit 2	Solve h system	15	52	9	21
	Set up s system	117	125	29	20
	Solve s system	15	68	14	21
Fully-	Set up h-s system	150	150	92	27
implicit 1	Solve h-s system	46	264	52	62
Fully-	Set up h-s system	225	223	138	28
implicit 2	Solve h-s system	46	264	52	62

Comparison of the five schemes (cont'd)

Scheme	# time steps	# Newton iterations	# CG iterations	# GMRES iterations
Fully-explicit	100	N/A	N/A	N/A
Semi-implicit 1	100	N/A	694	473
Semi-implicit 2	100	N/A	1215	897
Fully-implicit 1	100	300	N/A	5173
Fully-implicit 2	100	300	N/A	4438

Predicting computing time

Mesh size: 1700×1400, # time steps: 100

Scheme	Time	1 core	2 cores	4 cores	8 cores
Fully-explicit	T_A	13.19	6.91	3.65	1.48
	T_P	7.94	3.97	1.98	1.46
Semi-implicit 1	T_A	141.48	76.78	58.95	53.08
	T_P	90.74	45.82	36.34	38.74
Semi-implicit 2	T_A	281.71	151.18	106.71	99.52
	T_P	184.37	92.97	69.22	70.66
Fully-implicit 1	T_A	2411.12	1233.59	841.94	759.87
	T_P	1678.70	839.35	453.09	480.53
Fully-implicit 2	T_A	1988.75	1034.44	721.55	620.93
	T_P	1473.85	736.93	397.13	414.38

What about GPUs?

- More complex than predicting performance on CPUs
- Cost of data transfer between host (CPU) and device (GPU)
- Cost of data transfer between device global memory and local memory
  - can overlap with floating-point operations
- Device occupancy may not be 100%
- For a cluster of GPUs, cost of host-host communication must also be considered

Insufficient balance of FPs via bandwidth?

- Multicore CPU example: Intel Xeon quadcore E5504 2.0GHz
  - peak F=64.0 single-precision GFLOPS
  - peak B_M=19.2 GB/s
- GPU example: NVIDIA GeForce GTX 590
  - peak F=2488 single-precision GFLOPS
  - peak B_M=328 GB/s
- Memory-bandwidth bound code will suffer more on a GPU!

Test runs on Tianhe-1A

- **Tianhe-1A** — No.1 supercomputer on TOP500 in 2010, No.2 in 2011
  - 14,336 Intel X5670 6-core 2.93 GHz CPUs
  - 7,168 NVIDIA Tesla M2050 GPUs
  - Peak: 4.70 peta FLOPS
  - Linpack: 2.56 peta FLOPS
- Collaboration is being established between Simula and NUDT (developer of Tianhe-1A)
  - Test runs of basin-filling simulations
  - Successful collaboration depends on dedicated effort from both sides

Using CPUs on Tianhe-1A

Preliminary runs of the fully-explicit scheme on the CPUs of Tianhe-1A, using a 8000×8000 mesh

# CPU cores	GFLOPS
96	219.18
192	414.96
384	753.88
768	2272.33
1536	1961.21
3072	5254.15

Using GPUs on Tianhe-1A

Preliminary runs of the fully-explicit scheme on the GPUs of Tianhe-1A, using a 8000×8000 mesh

# GPUs	GFLOPS
4	174.09
8	280.47
16	542.35
32	828.94
64	1591.53
128	1855.36
256	3365.59

Mint: an automated translator from C to CUDA

To ease the pain of GPU programming...

- Input: C code with Mint pragmas

```
#pragma mint for nest(all) tile(16,16,1)
   for (int z=1; z<= k; z++)
    for (int y=1; y<= m; y++)
     for (int x=1; x<= n; x++)
      Unew[z][y][x] = c0 *  U[z][y][x] +
                   c1 * (U[z][y][x-1] + U[z][y][x+1] +
                   U[z][y-1][x] + U[z][y+1][x] +
                   U[z-1][y][x] + U[z+1][y][x]);
```

- Output: (auto-optimized) CUDA code
- Developed in collaboration between UCSD and Simula
- Mint can do extensive optimizations for stencil computations
  - Achieved about 80% performance of hand-coded and hand-optimized CUDA

Current and future activities of Mint

- Translation of real-world codes
  - AWP-ODC: 3D anelastic wave propagation in connection with earthquake simulations
  - 3D PMM: simulation of geological folding
  - 3D Harris interest point detection (computer visualization)
  - Basin-filling simulations
- Downloadable from https://sites.google.com/site/mintmodel/
- Future enrichment of Mint
  - Extension to multi-GPUs
  - Automated optimizations for non-stencil computations