Network Bandwidth & Minimum Efficient Problem Size

Paul R. Woodward
Laboratory for Computational Science & Engineering (LCSE), University of Minnesota

April 21, 2004

Build 3 virtual computers with Intel Pentium-Ms.

- Will analyze their performance on my gas dynamics application, PPM.
- Will demand the same 80% parallel efficiency on this application from each machine ($\Rightarrow N=128$)
- Easy to do this modeling exercise, since my laptop has a Pentium-M.
- Fix the CPU and the bandwidth to its local memory at the level of my laptop, so can concentrate instead on the interconnect.
- My laptop might not be your choice of CPU or of its packaging, but it nevertheless delivers very reasonable performance, as shown on next slides

Code kernel N=4096	Flops/ Cell Update (1-D pass)	Adds/ Cell Update (1-D pass)	Mults/ Cell Update (1-D pass)	Recips/ Cell Update (1-D pass)	Sqrts/ Cell Update (1-D pass)	Cvmgms /Cell Update (1-D pass)
SPPM scalar	701	255	264	23	8	68
SPPM vector	734	266	286	23	8	116
PPM SlowFlow	680	293	249	13	7	141
PPM MultiFluid	1434	620	647	39	8	373
PPM StateFair	1346	630	645	19	1	386

This table gives the computational makeup of 4 PPM code kernels used in various performance tests. The two versions of the sPPM benchmark kernel actually perform identical arithmetic. However, in the scalar implementation, a few redundant computations have been eliminated upon fusing multiple calls to the same subroutine in the vectorized version. The multifluid PPM kernel is by far the most complex. It contains several scalar loops dealing only with cells in strong shocks, and it has the largest temporary workspace that needs to be held in the cache memory. "Cvmgms" are vectorizable logic operations (conditional moves).

32-bit P-M	N = 32	N = 64	N = 128	N = 256	N = 512	N = 1024	N = 2048	N = 4096
Scalar sPPM	539	554	561	571	573	577	574	577
Vector sPPM	993	1247	1453	1588	1629	1598	1080	635
Slow-Flow PPM	1199	1321	1454	1474	1497	1484	1141	614
Multi-Fluid PPM	783	902	972	1007	252	223	386	480
State-Fair PPM	777	900	1002	1077	1091	965	668	546

Performance (Mflop/s) of PPM code kernels with 32-bit arithmetic on an Intel Pentium-M CPU running at 1.7 GHz in a Dell Inspiron 8600.

64-bit P-M	N = 32	N = 64	N = 128	N = 256	N = 512	N = 1024	N = 2048	N = 4096
Scalar sPPM	412	432	441	443	447	448	448	448
Vector sPPM	461	510	537	552	533	409	275	238
Slow-Flow PPM	613	652	673	677	662	532	312	266
Multi-Fluid PPM	491	528	549	530	379	176	226	218
State-Fair PPM	533	579	615	636	546	380	304	248

Performance (Mflop/s) of PPM code kernels with 64-bit arithmetic on an Intel Pentium-M CPU running at 1.7 GHz in a Dell Inspiron 8600.

32-bit 1.7 GHz PM	N = 32	N = 64	N = 128	N = 256	N = 512	N = 1024
Adds/Cell	3022	2052	1650	1469	1384	1343
Mults/Cell	3010	2047	1649	1470	1386	1345
Cvmgms/ Cell	1810	1234	996	889	838	814
Recips/ Cell	95.5	64.8	52.0	46.3	43.6	42.3
Sqrts/Cell	5.4	3.5	2.68	2.32	2.16	2.08
Flops/Cell	6401	4347	3493	3112	2931	2844
Sec/Δt	0.00762	0.0184	0.0551	0.205	0.829	3.52
Sec/Δt @N=4096	124.8	75.5	56.4	52.4	53.1	56.3
Mflop/s	645	724	779	747	695	636
Fraction 1D Perf.	83%	80%	78%	69%	64%	66%

Performance of StateFair PPM updating a full N×(3N/4) Grid Tile

64-bit 1.7 GHz PM	N = 32	N = 64	N = 128	N = 256	N = 512	N = 1024
Adds/Cell	3023	2052	1650	1469	1384	1343
Mults/Cell	3012	2049	1649	1470	1386	1345
Cvmgms/ Cell	1809	1235	996	889	838	814
Recips/ Cell	95.5	64.7	52.0	46.3	43.8	42.3
Sqrts/Cell	5.4	3.5	2.7	2.3	2.16	2.08
Flops/Cell	6397	4343	3492	3111	2931	2844
Sec/Δt	0.0114	0.0302	0.0993	0.384	1.60	7.16
Sec/Δt @N=4096	187	124	102	98.3	103	115
Mflop/s	430	442	432	398	359	316
Fraction 1D Perf.	81%	76%	70%	63%	66%	83%

Performance of StateFair PPM updating a full N×(3N/4) Grid Tile

Comments on the Preceding Performance Tables:

- The Pentium-M, unlike the Pentium-4 or the Itanium-2 CPUs from Intel, does not deliver a significant performance benefit for vector code using 64-bit arithmetic.
- The 1-D kernel tests are built to execute entirely out of the 1 MB cache, so long as the vector length allows this.
- The performance tables for StateFair PPM, which is similar in its flop and data access mix to MultiFluid PPM, show that the additional cost of data manipulations involving main memory are no more than 20% (32-bit) or 30% (64-bit).
- Thus local main memory bandwidth is no issue.

3 virtual interconnects:

- In the first computer, use today's COTS tech.
- In the second 2 computers, assume network bandwidth is what we need for the implementation of PPM that is given below.
- In all machines, local memory bandwidth is the same as in my Dell Inspiron 8600 laptop, namely 1.8 GB/sec (actual).
- Only difference is the network interconnect.
- If we demand 80% parallel efficiency, today's networks can run only gargantuan dynamically load balanced PPM runs.
- High bandwidth interconnect reduces the size of the smallest run I can do at this same efficiency.

Requirements Driving Programming Model:

- Dynamical load balancing assumptions for a static domain decomposition.
  - Time to execute tasks of a like type varies within a factor of 4 with cell-by-cell AMR.
  - Time to execute a task cannot be predicted with confidence to within 30% for AMR.
- Code simulates time dependent physical behavior in a spatial domain.
  - Highest level of parallelism achieved by spatial decomposition of the domain.
  - Different materials and conditions in different places at different times.
  - Impossibility of detailed load balance.

Fundamental Implications (more general than PPM):

- Update subdomains in self-scheduled loop in order to achieve load balance.
- Any computing element can update any domain, although it might prefer some domains to others.
- Signals from neighboring domains contained in ghost cell halos.
- Any necessary global knowledge communicated via iterations of subdomain updates.
- Coarse knowledge of distant regions communicated through coarse grid iterations (multigrid relaxation).
- Detailed knowledge of distant regions irrelevant

Fundamental Code Structure:

- The code has a hierarchical structure of tasks of increasing granularity.
  - Grid brick updates are largest tasks.
  - These consist of grid plane, pencil, or strip updates.
  - These consist of single cell updates, but these must be done by same CPU to allow vector performance enhancements.
  - Physics can be split into separate operations, but order is important.
- These tasks map naturally to a hierarchically organized computing system.

Fundamental Task Structure:

- Each type of task operates on a data context containing the relevant state of the grid cells to be updated (plus ghost cells).
- Each task uses a temporary workspace held in a temporary local memory.
- Each task can run without interruption or communication using only this workspace, once the data context has been copied into it.
- If the next task is known and its data is ready, its data context can be prefetched while the present task executes.
- Can write back results of task while next task executes.

Data Contexts:

- These could be sections of some global array or set of global arrays.
- These could be named data objects, or "files."
- Either way, they must be shared.
- They can be mirrored for fault tolerance.
- Prefetch data context by series of asynchronous read or DMA operations.
  - Express as assignment of global array sections to local arrays.
  - Or express as asynchronous read, with offset and length.
  - Either way, MUST be asynchronous.

Smallest subdomain Updates:

- These apply physics operators to local data in the temporary workspace.
- They can be written for a single CPU.
- They are naturally written by physicists.

Larger subdomain Updates:

- These operate only on local data in the temporary workspace.
- They can be written for multiprocessor systems.
- All they do is call smaller subdomain updates.
- Hence they just shuffle data around, but do not work on that data directly.
- They can be written without knowledge of physics

Implications of the Programming Model for the Machine Design:

- Smallest computing element is a CPU (a Pentium-M, just like in my laptop).
  - CPUs update the smallest subdomains.
- A group of 32 CPUs will be called an SMP, whether or not it has a shared memory.
  - SMPs update larger subdomains, which are collections of the smaller subdomains updated by the CPUs in the SMP.
- A group of 32 SMPs will be called a cluster.
  - Clusters update large subdomains, which are collections of those updated by SMPs.
- A machine consists of 32 clusters.

Assume a Hierarchical arrangement of Crossbars:

- Ideal network topology.
- Permits dynamic load balancing.
- Easy to program.
  - Code has hierarchy of tasks of increasing granularity.
  - Map tasks to hierarchical computing system.
  - Each task operates on data context and uses temporary workspace in a cache or local memory.
  - Triple buffering of data at each level.
- Computer big enough to be interesting: 32,768 CPUs (32×32×32).

Possible Mappings of Tasks to Computing Elements and their Costs:

- Inevitable load imbalances demand that, for efficient computation, at any level of the computing hierarchy I must map several times as many tasks of like type as I have computing elements at that level.
- The smaller the task that I can efficiently map onto the largest computing element, the smaller the problem that I will be able to efficiently run on the system.
- For the purposes of this exercise, we will assume 32 computing elements (hence 128 tasks) combining at each level to form the next hierarchical level.

Multifluid PPM Tasks:

- The table on the next slide lists a number of tasks into which the multifluid PPM code can be decomposed.
  - The larger the task, the greater the data reuse and hence the lower the network bandwidth required to feed it with data contexts.
  - The larger the task, the larger the required temporary workspace (cache or other local memory).
  - As the workspace size increases over 3 ½ orders of magnitude, the bandwidth requirement decreases over 2 orders of magnitude, & latency requirement (≈0.1% of task completion time) is relaxed by 6 orders of magnitude.

PPMMF Task Hierarchy for 1 CPU, Strip Length = 64:

Task	Bytes/ flop	Flops	Bytes in+out	Work space	Time (sec)	MB /sec
Loop 1	8.00	1088	8704	5632	2.63 μ	3156
Loop 2	4.80	1280	6144	7680	2.30 μ	2548
Loop 3	0.88	5824	5120	7680	9.24 μ	528
PPB6	0.46	17,792	8192	22,016	27.0 μ	289
1-D Strip	1.12	100 K	119 K	267 K	173 μ	703
1-D Pencil	0.344	6.42 M	2.50 M	2.64M	14.5 m	145
1-D Brick	0.239	411 M	93.6 M	97 M	0.930	101
3-D Brick	0.0785	1.74 G	130 M	181 M	3.94	33.1
128^3 3-D Brick 2Δt	0.0392	27.9 G	1.04 G	1.45 G	63.1	16.5

PPMMF Task Hierarchy

Seek a hierarchy of tasks such that:

1) Build each larger task out of 128 smaller ones (so each computing element does, on average, 4 tasks).
2) Either a 1-D or a 3-D brick update is one of the tasks.

If a single strip 1-D update is a task, then individual vectorizable loops cannot be considered as tasks, since the loops that comprise a strip update must be done in order & cannot be done in parallel.

Use only 3-level task hierarchy.

Demand a constant level of parallel efficiency:

efficiency = % non-redundant computation

$= (N / (N+10))^3$

For $N = 128$, are 80% efficient.

Task	Bytes/ flop	Flops	Bytes in+out	Work space	Time (sec)	MB /sec
3-D Brick (128^3)	0.0756	11.3 G	815 MB	1048 MB	25.6 sec	31.8 MB/s
128 3-D Bricks	0.0756	1.45 T	102 GB	134 GB	1.71 min	1.99 GB/s
128^2 3-D Bricks	0.0756	185 T	13.1 TB	17.2 TB	6.84 min	63.6 GB/s
128^3 3-D Bricks	0.0756	23.7 P	1.63 PB	2.18 PB	27.4 min	1.99 TB/s
Run of 163,840 Δt	0.0756	3883 Exa-flops	16384^3 grid		8.54 years	GigE, JB4x, JB64x

This conservative task mapping, typical of PPM runs on today's computing systems, demands an 8.5 year run in this model if we require 80% efficiency.

Task	Bytes/ flop	Flops	Bytes in+out	Work space	Time (sec)	MB /sec
1-D Pencil (8×16)	0.311	24.6 M	7.29 MB	18.7 MB	55.6 msec	131 MB/s
3-D Brick (128^3)	0.0756	11.3 G	815 MB	1048 MB	0.222 sec	1.99 GB/s
128 3-D Bricks	0.0756	1.45 T	102 GB	134 GB	0.890 sec	63.6 GB/s
128^2 3-D Bricks	0.0756	185 T	13.1 TB	17.2 TB	3.56 sec	1.99 TB/s
Single Run of 40,960 Δt	0.0756	7.58 Exa-flops	2048×4096^2 grid		1.69 days	Myri, JB4x, JB64x

If we provide 4 times the bandwidth to each CPU, we achieve the same 80% efficiency with a run only half the size of the largest run ever done on the Earth Simulator.

Task	Bytes/ flop	Flops	Bytes in+out	Work space	Time (sec)	MB /sec
1-D Strip (128)	1.16	192 K	222 K	365 KB	332 µsec	638 MB/s
1-D Pencil (8×16)	0.311	24.6 M	7.29 MB	18.7 MB	1.74 msec	8.18 GB/s
3-D Brick (128^3)	0.0756	11.3 G	815 MB	1048 MB	25.0 msec	63.6 GB/s
128 3-D Bricks	0.0756	1.45 T	102 GB	134 GB	100 msec	1.99 TB/s
Single Run of 10,240 Δt	0.0756	14.8 Peta-flops	1024×512^2 grid		17.1 mins	JB, JB12x, JB64x

If we provide 20 times the bandwidth to each CPU and 4 times the bandwidth to each SMP, we achieve the same 80% efficiency with a modest run, finished in a quarter hour.

For each Virtual Computing System:

- Same 80% parallel efficiency, resulting from use of 128^3 3-D grid brick updates.
- Same 14.5 Tflop/s delecivered performance.
- Totally different user experiences.

For the High-Bandwidth Machine:

- Time sharing makes sense, since one can get something useful done in 15 minutes without running at such low efficiency that the resource would be wasted.
- Small, exploratory runs could be done on this system rather than requiring additional, smaller cluster systems for this purpose.

Remember the Assumptions We Made at the Outset:

- This analysis applies to runs that demand dynamic load balancing, and it assumes that we achieve this via simple self-scheduled task loops.
- Some very regular problems, such as homogeneous turbulence, could easily be run in pure SIMD mode, as on MPPs like the T3E.
  - Could achieve perfect static load balance.
  - On first machine, could put 1 brick on each CPU, so that need have only 32^3 rather than 128^3 bricks, and run would take only 12.2 days
  - 2nd machine could have 1 brick per 32 CPUs, so 8×8×16 bricks, and run takes only 4.6 hours.
  - 3rd machine has 32 bricks, run takes 2.1 minutes.

The following slides give details used to get some of the numbers in the tables.

1-D 128-long Grid Strip Update:

- For 128-cell PPMMF 1-D update, have 160×(1+9×(140/128)) B/cell and 1434×(134/128) flops/cell, using 64-bit arithmetic. This is 1735/1501.22 = 1.16 B/flop.
- Laptop CPU performance is 579 Mflop/s.
- The time required to perform one of these little tasks is just 1434 × (128+6) / 579 µsec = 332 µsec.
- This requires a memory bandwidth, with triple buffering of the data context, of 1735×128 / 331.9 B / µsec = 638 MB/sec.
- Data context is 1600 × 76 B = 119 KB.
- Workspace ≈ 217 KB + 250×8×76 B = 365 KB.

1-D 8×16×128 Grid Pencil Update:

- For 128-cell PPMMF 1-D update, have 160×(1+1.75×(140/128)) B/cell and 1434×(134/128) flops/cell, using 64-bit arithmetic. This is 466.25/1501.22 = 0.311 B/flop.
- Laptop CPU performance is 442 Mflop/s.
- The time required to perform one of these tasks is just 1434 × (128+6) × 128 / 442 µsec = 55.6 msec.
- This requires a memory bandwidth, with triple buffering of the data context, of 466.25×128×128/55647 B/µsec = 131 MB/sec.
- Data context is 160×1.75×140×128 B = 4.79 MB.
- Workspace is about 7.29 MB + 32×365 KB = 18.7 MB.

3-D 128^3 Brick Update:

- For 128^3 PPMMF 3-D update, have 160 × $(1+(148/128)^3)$ B/cell & $3\times1434\times(138/128)^3$ flops/cell, using 64-bit arithmetic. This is 407.33/5391 = 0.0756 B/flop.
- Laptop CPU performance is 442 Mflop/s.
- The time required to perform one of these little tasks is just $3\times1434 \times 138^3 / 442$ µsec = 25.6 sec.
- This requires a memory bandwidth, with triple buffering of the data context, of 407.33×128^3 / 25.6 B/sec = 31.8 MB/sec.
- Data context is 160 × 148^3 B = 495 MB.
- Workspace is about 815 MB + 32×7.29 MB

3-D 128^3 Brick Double Update:

- For 128^3 PPMMF double 3-D update, have $160 \times (1+(168/128)^3)$ B/cell & $6 \times 1434 \times (148/128)^3$ flops/cell, using 64-bit arithmetic. This is 521.8/13300 = 0.0392 B/flop.
- Laptop CPU performance is 442 Mflop/s.
- The time required to perform one of these little tasks is just $6 \times 1434 \times 148^3 / 442$ μsec = 63.1 sec.
- This requires a memory bandwidth, with triple buffering of the data context, of $160 \times (168^3+128^3) / 63.1$ B/sec = 16.5 MB/sec.
- Data context is 160×168^3 B = 724 MB.
- Workspace is about 2×724 MB = 1.45 GB.