Imperial College London

From here to there

challenges for ~~peta~~ exa-scale transient simulation

Lawrence Mitchell[1,*]

27th March 2017

[1]Departments of Computing and Mathematics, Imperial College London
*lawrence.mitchell@imperial.ac.uk

What do we want?

50GHz single-core CPUs

What do we have?

Too many cores

Chip	Cores	TF/s	GB/s	F/B	Power
NVidia P100	56 (3584)	5.3	730	7.2	250 (21 GF/W)
Xeon Phi 7290F	72	3.5	450 + 100	5.4	260 (13 GF/W)
Broadwell	22	0.78	150	5.2	140 (5.6 GF/W)

Exascale

- 190K NVidia P100s, 1e9-way concurrency, 150MW
- 290K Intel Phis, 1e8-way concurrency, 220MW
- 1.3M Intel Broadwells, 3e7-way concurrency, 540MW

Too many cores

Chip	Cores	TF/s	GB/s	F/B	Power
NVidia P100	56 (3584)	5.3	730	7.2	250 (21 GF/W)
Xeon Phi 7290F	72	3.5	450 + 100	5.4	260 (13 GF/W)
Broadwell	22	0.78	150	5.2	140 (5.6 GF/W)

Exascale

- 190K NVidia P100s, 1e9-way concurrency, 150MW
- 290K Intel Phis, 1e8-way concurrency, 220MW
- 1.3M Intel Broadwells, 3e7-way concurrency, 540MW
- 1 Boeing 747, 140MW

Too many cores

Chip	Cores	TF/s	GB/s	F/B	Power
NVidia P100	56 (3584)	5.3	730	7.2	250 (21 GF/W)
Xeon Phi 7290F	72	3.5	450 + 100	5.4	260 (13 GF/W)
Broadwell	22	0.78	150	5.2	140 (5.6 GF/W)

Exascale

- 190K NVidia P100s, 1e9-way concurrency, 150MW
- 290K Intel Phis, 1e8-way concurrency, 220MW
- 1.3M Intel Broadwells, 3e7-way concurrency, 540MW
- 1 Boeing 747, 140MW
- 1 Google, 650MW

Too many cores

Chip	Cores	TF/s	GB/s	F/B	Power
NVidia P100	56 (3584)	5.3	730	7.2	250 (21 GF/W)
Xeon Phi 7290F	72	3.5	450 + 100	5.4	260 (13 GF/W)
Broadwell	22	0.78	150	5.2	140 (5.6 GF/W)

Exascale

- 190K NVidia P100s, 1e9-way concurrency, 150MW
- 290K Intel Phis, 1e8-way concurrency, 220MW
- 1.3M Intel Broadwells, 3e7-way concurrency, 540MW
- 1 Boeing 747, 140MW
- 1 Google, 650MW
- 1 Sizewell B, 1200MW

Too many cores

Chip	Cores	TF/s	GB/s	F/B	Power
NVidia P100	56 (3584)	5.3	730	7.2	250 (21 GF/W)
Xeon Phi 7290F	72	3.5	450 + 100	5.4	260 (13 GF/W)
Broadwell	22	0.78	150	5.2	140 (5.6 GF/W)

Exascale

- 190K NVidia P100s, 1e9-way concurrency, 150MW
- 290K Intel Phis, 1e8-way concurrency, 220MW
- 1.3M Intel Broadwells, 3e7-way concurrency, 540MW
- 1 Boeing 747, 140MW
- 1 Google, 650MW
- 1 Sizewell B, 1200MW
- 1 UK, 35GW

What's happened to the chips

- Number of transistors still increasing exponentially
- Frequency flat since c. 2005
- Performance through on-chip parallelism: "now it's your problem"
- Wider "atomic" floating point instructions

Chip	Cores	Clock	Vector width	Historical proxy
P100	56	1.5	32	CM-1
Broadwell	22	2.2	4	
Phi	72	1.5	8	Cray X1
Skylake	32	2.2	8	
ARMv8 (Cavium)	54?	2?	4-32?	

What should we do?

Run LINPACK

But seriously

- High arithmetic intensity (flops are cheaper than bytes)
- Vectorise, vectorise, vectorise (only way to achieve flops)
- Avoid bulk synchronous computation (performance resilience)
- Reduce and/or amortise communication (hide latency)

The status quo

- ✗ Low order, memory bound
- ✗ Vectorisation left to compiler (?)
- ✗ Iterative schemes with blocking reductions
- ✗ Simple communication patterns (not optimal?)

What to look for

Algorithmic optimality

Notation

N – total number of degrees of freedom;

P – total number of processes;

$T(N, P)$ – time to solution.

Desired

$\mathcal{O}(N)$ computational complexity;

$\mathcal{O}(\log P)$ communication complexity.

Be aware of the constants!

Parallel scalability

Weak scaling

Constant local work N/P.

Scalable code has $T(N, P) = T(2N, 2P)$.

Strong scaling

Decreasing local work N/P.

Scalable code has $T(N, P) = 2T(N, 2P)$.

Transient simulation

Time-resolved transient simulations do not weak scale. Sad!

What to do?

- Get algorithmics right
- Work hard to attack constant factors
- Work on strong scaling efficiency
- Develop *predictive* models of performance to *understand* why codes behave how they do.

Predicting scalability

Summarising Fischer, Heisey, and Min (2015).

Notation

- parallelisable work: $T_a(N,P) = T_a(N,1)/P$
- communication: $T_c(N,P)$
- serial overhead: $c \approx 0$
- time to solution

$$T(N,P) = \begin{cases} T_a(N,P) + T_c(N,P) + c & \text{synchronous} \\ \max(T_a(N,P), T_c(N,P)) + c & \text{asynchronous} \end{cases}$$

- scaling efficiency: $\eta = \frac{T(N,1)}{PT(N,P)}$

When to stop adding cores?

Minimum $T(N,P)$

Find P such that $\frac{\mathrm{d}T(N,P)}{\mathrm{d}P} = 0$.

Typically too expensive (wasting many core hours).

A compromise

Find P such that $T_a(N,P) = T_c(N,P)$, $\eta = 0.5$ for synchronous case.

Theorem (Anonymous)

Krylov methods strong scale to $N/P \approx 30000$.

Explicit schemes are a little better $N/P \approx 10000$.

"Reductions limit scalability"

Empirical scaling limits

- Measure $T(N, P_{\text{min}})$ and $T(N, P)$ for a range of process counts.
- Pick P_{opt} such that $P_{\text{opt}}T(N, P_{\text{opt}}) = 2T(N, P_{\text{min}})$.
- How do I know if that is any good?

Building blocks

Computation

Measure S, e.g., flops with $P = 1$, N large.

"atomic" unit of computation takes time $t_a = S^{-1}$.

Communication

Linear model, latency + bandwidth.

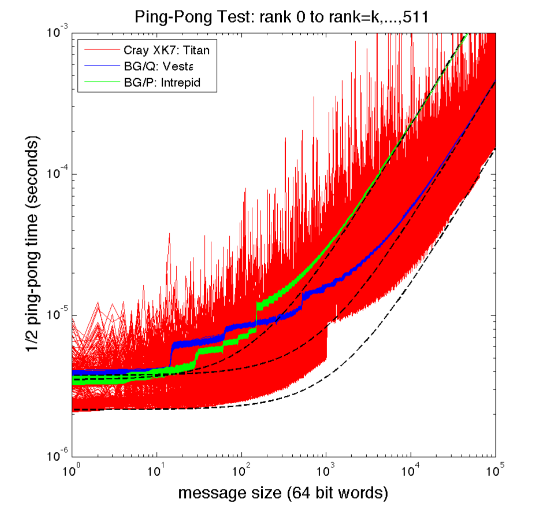
Time (s) to send m doubles

$$t_c(m) = \alpha^* + \beta^* m$$

non-dimensionalise, $\alpha = \alpha^*/t_a$, $\beta = \beta^*/t_a$.

$$t_c(m) = (\alpha + \beta m)t_a$$

Run some measurements



From Fischer, Heisey, and Min (2015).

Performance variability

- Model is pretty good
- Network topology + load can affect even simple codes
- BlueGene has torus network, each job gets a convex subset
- Not true on Cray (Dragonfly), network traffic from other jobs can affect your performance (Prisacari et al. 2014).

Jacobi iteration, 7-point 3D stencil

$$u_i^{k+1} = a_{ii}^{-1}\left(f_i + \sum_{j\neq i} a_{ij}u_j^k\right)$$

counting operations with N/P entries per process.

$$T_a = 14(N/P)t_a.$$

With a block decomposition, each face exchange moves $(N/P)^{2/3}$ values, so

$$T_c = 6\left(\alpha + \beta(N/P)^{2/3}\right)t_a.$$

With $\alpha = 3750, \beta = 2.86$ (BG/Q), $T_a = T_c$ when $N/P \approx 1700$. *Independent* of P.

If $\beta = 0$, $N/P \approx 1600$.

Conjugate gradients, 7-point 3D stencil

$$T_a = 27(N/P)t_a$$

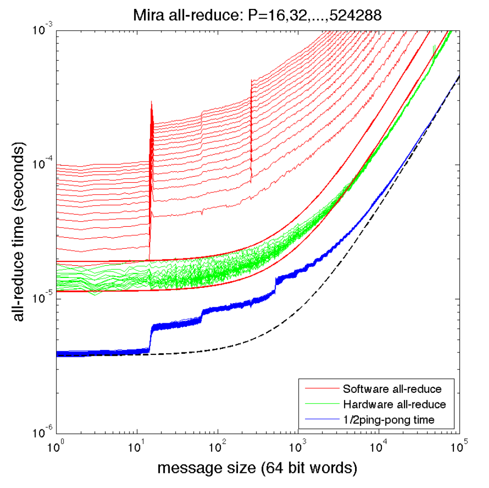
Again, we need 6 face exchanges, plus two reductions (each $2\alpha t_a \log_2 P$)

$$T_c = 6\left(\alpha + \beta(N/P)^{2/3}\right) t_a + 2 \cdot 2\alpha t_a \log_2 P.$$

Now the scaling limit is P-dependent.

- $P = 10^6$: $N/P \approx 12000$;
- $P = 10^9$: $N/P \approx 17000$.

But wait



From Fischer, Heisey, and Min (2015).

- Hardware-level allreduce on BlueGene is *P independent*.
- On the full machine, a reduction costs 5α.

Round again

$$T_c = 6\left(\alpha + \beta(N/P)^{2/3}\right) t_a + \cancel{2 \cdot 2 \log_2 P \alpha t_a} + 2 \cdot 5 \alpha t_a.$$

Now we have *P-independent* scaling behaviour, $N/P \approx 2100$.

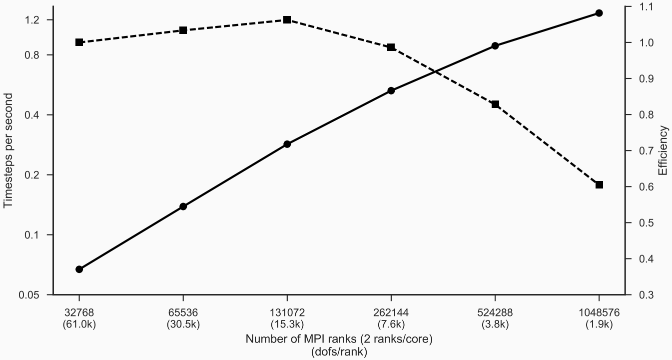
Using only a single reduction, we can get to $N/P \approx 1500$.

8x more strong scaling on $P = 10^9$, with no increase in power consumption.

A similar analysis can be done for multilevel algorithms, e.g. for Poisson $N/P \approx 10000$ (constant reduction complexity).

Some data points

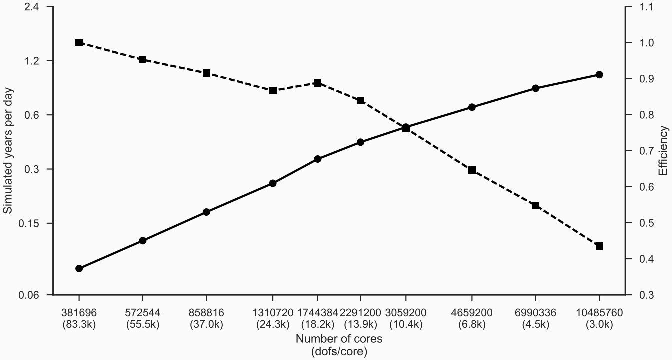
3-D incompressible Navier-Stokes for reactor cooling, NEK5000. High order, spectral element. 60% time in multigrid Poisson solves.



Data reproduced from Fischer, Heisey, and Min (2015).

Some data points

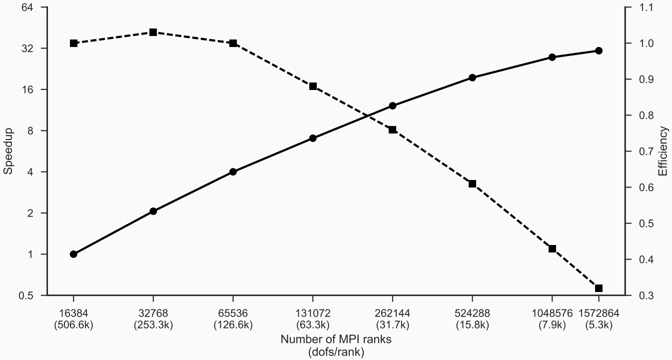
3-D non-hydrostatic baroclinic instability 3km resolution, Gordon Bell prize 2016. Low order, finite volume. Most time in multigrid Helmholtz solve.



Data reproduced from Yang et al. (2016).

Some data points

3-D nonlinear Stokes for mantle convection, Gordon Bell prize 2015. High order, finite element. Time split between viscous and pressure-Poisson multigrid solves.



Data reproduced from Rudi et al. (2015).

Latency hurts

- When strong-scaling mesh codes, you don't care about network bandwidth.
- Decreasing α is important, pester your vendor!
- Faster cores (relative to network) means worse strong scaling.
- Faster code means worse strong scaling.

Some thoughts on climate codes

Conjecture

Operational climate models make nowhere near optimal use of current hardware.

Extrapolating current SYPD to larger problems is perhaps not useful, unless we think the current models are good.

- More work means scaling should improve.
- Will column-wise data decomposition start to hurt?
- Lobby for power spend on interconnect, not cores?
- Don’t forget to focus on minimising time-to-solution first.

What might we do?

Improving SYPD

- Better serial performance. Is it the case that current codes make efficient use of hardware?
- High order? Only useful if we can use fewer dofs. Are models in the asymptotic region where we expect exponential accuracy gains from high order discretisations?
- Better strong scaling. Necessary to counteract timestep restrictions with increasing resolution.

Better serial performance

- Ground up rewrites of models?
- Optimising “line by line” doesn’t work, we’re stuck in local minima. e.g. changes in data layout require a large scale changes if the data model is implicit.
- Look for opportunities to reduce algorithmic complexities
- Yang et al. (2016) and Rudi et al. (2015) are examples of what you can do for single components.

High order?

- High order, flop heavy, schemes are more suited to modern architectures
- But often not in asymptotic convergence region
- Need to have competitive performance *per dof*
- FE probably preferable to FV or FD, since minimal stencil (less comms).

Addressing latency

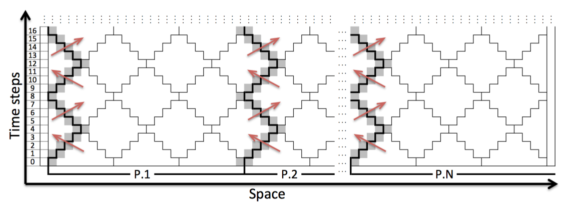
- Reducing α has a big effect on scaling limits
- $\alpha \to \alpha/10$ would allow scaling Poisson multigrid to $N/P \approx 900$. 10x in time to solution for same power.
- Similarly, hardware reductions are *really* important.
- Would we be happy if vendors spent more of the power budget on network and less on chips?

τ-FAS

- τ formulation of multigrid (Brandt (1977), Brandt and Livne (2011, §8.3)) admits low data transfer implementation (Brandt and Diskin 1994).
- Performance modelling and results for 27 point FV Poisson problem in Adams et al. (2016).
- Worthwhile to try if you already have a FAS for your problem?

Tiling to amortise latency

- *Diamond tiling* is a well known optimisation in computer science for stencil codes.



- Typically used for better cache usage.
- Can be extended to hide network latency.
- Good analysis in Malas et al. (2015)
- "Rediscovered" in Alhubail and Wang (2016).
- Explicit schemes only.

Asynchronous algorithms

- Harden against OS jitter by removing barriers
- Hide latency
- Potential for soft error recovery
- Is MTTF *really* a problem? The same things were being warned of petascale systems.

Asynchronous algorithms

- Harden against OS jitter by removing barriers
- Hide latency
- Potential for soft error recovery
- Is MTTF *really* a problem? The same things were being warned of petascale systems.

Pipelined Krylov methods

- Use asynchronous reductions Ghysels et al. (2013).
- Not aware of any group other than Vanroose's that shows such performance improvements.
- Best suited to simple preconditioners.

- Harden against OS jitter by removing barriers
- Hide latency
- Potential for soft error recovery
- Is MTTF *really* a problem? The same things were being warned of petascale systems.

s-step Krylov methods

- AKA communication avoiding Krylov.
- Mostly work from Demmel's group.
- Again, don't work with "good" preconditioners.
- Erin Carson's thesis (Carson 2015) is an excellent, and honest, summary of the current state.

Is time parallel the answer?

- At some point, traditional timestepping will stop scaling
- Time parallel is perhaps a way around this
- Need to be honest. Can we get speedups relative to the best "traditional" model?
- Are we better off running bigger ensembles? Better data assimilation?

Questions?

References I

Adams, M. F. et al. (2016). "Segmental Refinement: A Multigrid Technique for Data Locality". *SIAM Journal on Scientific Computing* 38. doi:10.1137/140975127. arXiv: 1406.7808 [ma.NA].

Alhubail, M. M. and Q. Wang (2016). "The swept rule for breaking the latency barrier in time advancing PDEs". *Journal of Computational Physics* 307. doi:10.1016/j.jcp.2015.11.026. arXiv: 1504.01380 [cs.CE].

Brandt, A. and O. Livne (2011). *Multigrid Techniques*. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611970753.

Brandt, A. (1977). "Multi-level adaptive solutions to boundary-value problems". *Mathematics of Computation* 31.

Brandt, A. and B. Diskin (1994). "Multigrid solvers on decomposed domains". *Contemporary Mathematics* 157.

References II

Carson, E. C. (2015). "Communication-Avoiding Krylov Subspace Methods in Theory and Practice". PhD thesis. University of California, Berkeley.

Fischer, P. F., K. Heisey, and M. Min (2015). *Scaling limits for PDE-based simulation*. Tech. rep. Argonne National Laboratory. doi:`10.2514/6.2015-3049`.

Ghysels, P. et al. (2013). "Hiding Global Communication Latency in the GMRES Algorithm on Massively Parallel Machines". *SIAM Journal on Scientific Computing* 35. doi:`10.1137/12086563X`.

Malas, T. et al. (2015). "Multicore-Optimized Wavefront Diamond Blocking for Optimizing Stencil Updates". *SIAM Journal on Scientific Computing* 37. doi:`10.1137/140991133`.

References III

Prisacari, B. et al. (2014). “Efficient Task Placement and Routing in Dragonfly Networks”. *Proceedings of the 23rd ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC’14).* Vancouver, Canada. doi:`10.1145/2600212.2600225`.

Rudi, J. et al. (2015). “An Extreme-scale Implicit Solver for Complex PDEs: Highly Heterogeneous Flow in Earth’s Mantle”. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC ’15. Austin, Texas: ACM. doi:`10.1145/2807591.2807675`.

Yang, C. et al. (2016). “10M-core Scalable Fully-implicit Solver for Nonhydrostatic Atmospheric Dynamics”. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC ’16. Salt Lake City, Utah: IEEE Press.