Abstract

While we speculate what exascale hardware might look like, state-of-the-art numerics and our machines already diverge. Many new hardware generations or ingredients such as Skylake, manycores or Intel's Optane reduce caches per core, make more and more cores share one interconnect, or introduce additional memory levels with high latency. At the same time, many modern algorithmic paradigms such as multigrid, particle-in-cell or predictor-corrector schemes require irregular, non-continuous, repeated memory accesses, as well as non-trivial (meta) data assembly. As a result, data assembly, movement and exchange become constraining factors when we upscale or tune scientific software. We have to avoid them.

In this talk, we generalise the term communication-avoiding. We make it comprise (i) the reduction of data volume, (ii) the elimination of (meta) data generation, (iii) the reduction of data exchange frequency, (iv) the homogenisation of data access, (v) data access hiding and (vi) the localisation of data transfers. These criteria apply to both classic data exchange between compute nodes as well as data movements on the chip.

Communication-avoiding then tackles the problematic divergence sketched above. While every code might require tailored solutions of its own to become communication-avoiding, we present some algorithmic techniques - for multigrid, particle-in-cell and predictor-corrector schemes - which seem to be generic patterns. They can inspire us how to write communication-avoiding software for various applications.

Stop talking to me - some communication-avoiding programming patterns

IPAM Workshop IV: New Architectures and Algorithms

D.E. Charrier, B. Hazelwood, B. Reps, B. Verleye, M. Weinzierl, me and many others

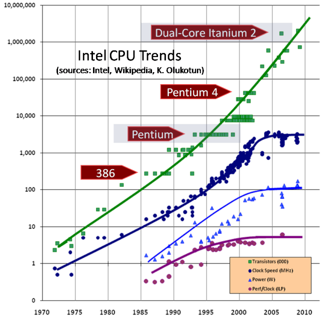
Durham Cathedral, www.dur.ac.uk

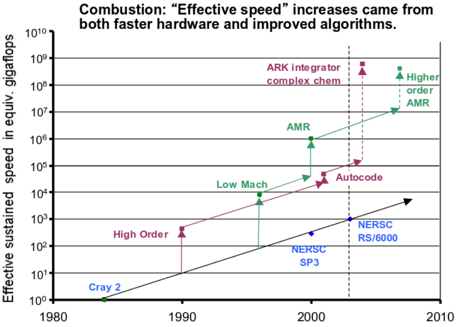
Why I love to think about algorithms and their implementation . . .

Chris Johnson (SCI, Utah):
Before the great discovery was the creation of a new tool!

("The Golden Age of Computing")

Performance increases—tool improves





H. Sutter: The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software, 2005 (left)

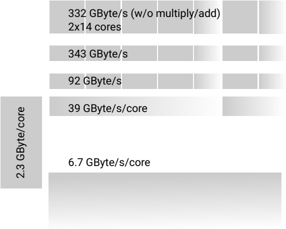
D. Keyes: SCaLeS Report, Vol. 2, 2004 (right)

- Both sides improve our tools, yet are not orthogonal
- Better algorithms drive machine evolution(*)
- Changing machine characteristics require algorithmic rethinking
- Don't be naïve: hardware poses challenges to algorithms

(*)If you believe in co-design:

`https://www.hpcwire.com/2016/07/12/isc-workshop-tackles-co-developments-thorny-challenges`

This talk's challenge: data movements become showstopper



Left: Node of SuperMUC Phase 2; right: SuperMUC ©IBM

Vertical New memory layers (Optane), new cache modi (Skylake)

Horizontal More cores per node, cache and network competition

The next big jumps on the algorithm side

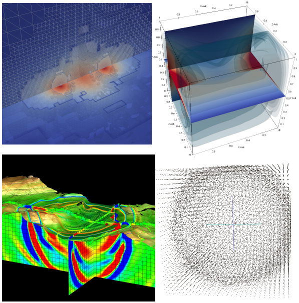
Remove data movement dependencies both intra-node & inter-node

= Remove communication dependencies between computer components

= Communication-avoiding algorithms

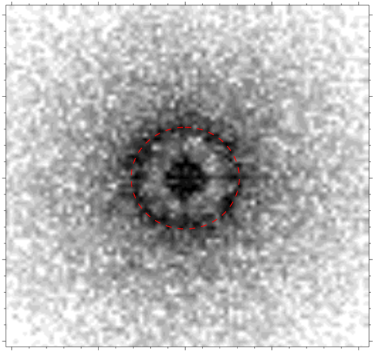
but don't give up on the cool mathematics/algorithms

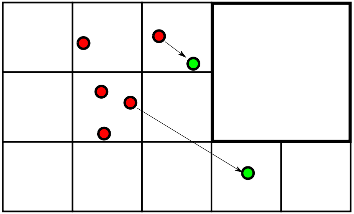
Outline



- Motivation
- Demonstrators
- Communication-avoiding techniques
  - Avoid assembling data
  - Give up on task-traversal association
  - Prioritise communication tasks
  - Predict synchronisation
  - Drop the IEEE standard
- Wrap-up

Particle-In-Cell (PIC)





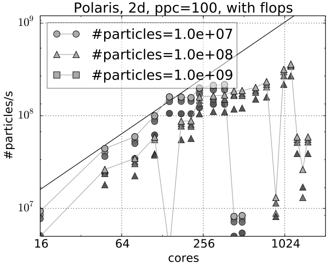
1. Solve PDE on (dynamically adaptive) mesh
2. Interpolate PDE solution onto particles
3. Move particles (suprathermal)
4. Restrict to PDE's rhs

Weinzierl, T., Verleye, B., Henri, P., Roose, D. *Two Particle-in-Grid realizations on Spacetrees*. Parallel Computing 52, 42–64, 2016. arXiv:1508.02435

Weinzierl, T. *The Peano software—parallel, automaton-based, dynamically adaptive grid traversals*. ACM Transaction on Mathematical Software (TOMS), 2nd revision. arXiv:1506.04496

Eckhardt, W., Glas, R., Korzh, D., Wallner, S., Weinzierl, T. (2016), *On-the-fly memory compression for multibody algorithms*, in Joubert, G.R., Leather, H., Parsons, M., Peters, F., Sawyer, M. eds, Advances in Parallel Computing 27: International Conference on Parallel Computing (ParCo) 2015. Edinburgh, Scotland, IOS Press, Amsterdam, 421–430

PIC



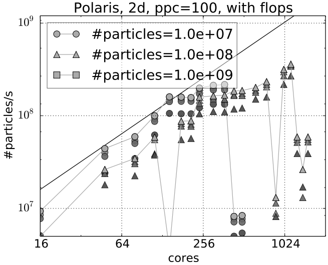
Likes

- Lagrangian plus Eulerian (best fit per subproblem)
- Break (Cartesian) mesh constraints
- Keep Cartesian efficiency for PDE

Flaws

- Particle updates/integrator cheap (arithmetic intensity)
- Maintain mapping
- Limited scalability
- Speed drops

PIC



Likes

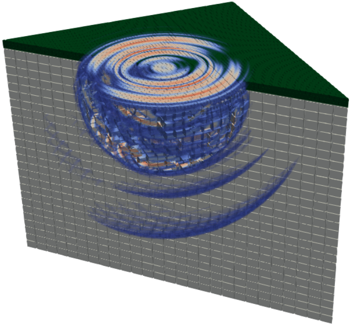
- Lagrangian plus Eulerian (best fit per subproblem)
- Break (Cartesian) mesh constraints
- Keep Cartesian efficiency for PDE

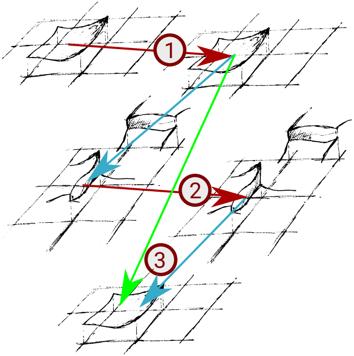
Flaws

- Particle updates/integrator cheap (arithmetic intensity)
- Maintain mapping
- Limited scalability
- Speed drops

Speed drops result from latency-sensitiveness of bandwidth-bound particle movers

ADER-DG

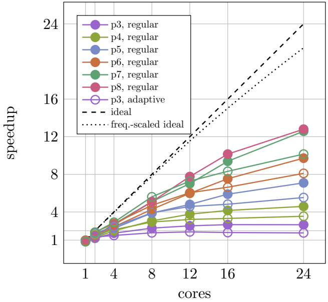


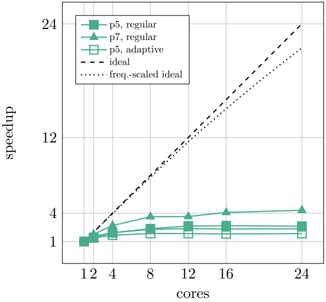


1. Predictor (non-linear iterate, implicit space-time)
2. Riemann
3. Corrector

Weinzierl, T. *The Peano software—parallel, automaton-based, dynamically adaptive grid traversals*. ACM Transaction on Mathematical Software (TOMS), 2nd revision. arXiv:1506.04496
Charrier E. D., Weinzierl T. *Stop talking to me – a communication-avoiding ADER-DG realisation*. arXiv:1801.08682
Dumbser M., Fambri F., Tavelli M., Bader M., Weinzierl T. *Efficient implementation of ADER discontinuous Galerkin schemes for a scalable hyperbolic PDE engine*. arXiv:1808.03788

ADER-DG





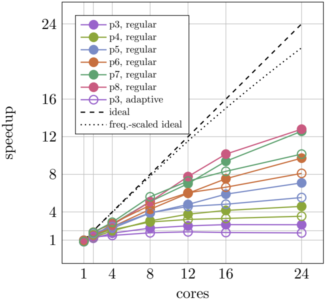
Likes

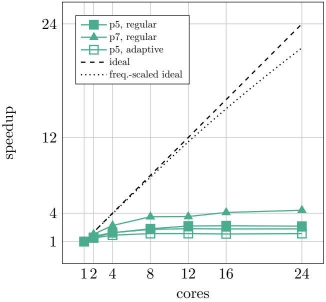
- Single shot (for large *p*)
- Localisation with expensive STP (arithmetic intensity)

Flaws

- Three phases (synchronisation and data loads)
- Two phases extremely cheap (arithmetic intensity)

ADER-DG





Likes

- Single shot (for large *p*)
- Localisation with expensive STP (arithmetic intensity)

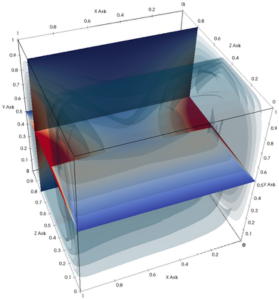
Flaws

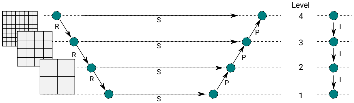
- Three phases (synchronisation and data loads)
- Two phases extremely cheap (arithmetic intensity)

Obtained scalability suffers from one bandwidth-bound algorithm phases, time step size synchronisation and dynamic AMR's consistency constraints

(Additive) Multigrid, BPX, . . .

with complex shift

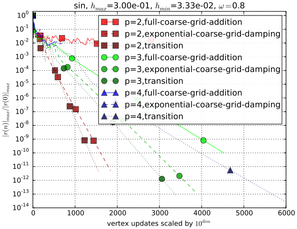


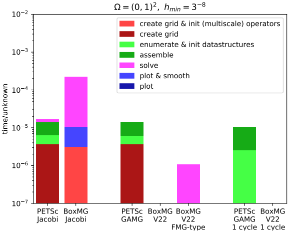


1. Solve on fine level
2. Solve correction on coarser level
3. Prolongate correction and sum up on fine level

Reps, B., Weinzierl, T. *Complex additive geometric multilevel solvers for Helmholtz equations on spacetrees*. ACM Transactions on Mathematical Software (TOMS), 44(1), 2:1–2:36, 2017. arXiv:1508.03954
Weinzierl, M., Weinzierl, T. Quasi-matrix-free hybrid multigrid on dynamically adaptive Cartesian grids, ACM Transactions on Mathematical Software (TOMS), 44(3), 32:1–32:44, 2018. arXiv:1607.00648

(Additive) Multigrid, BPX, . . .





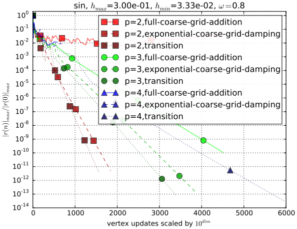
Likes

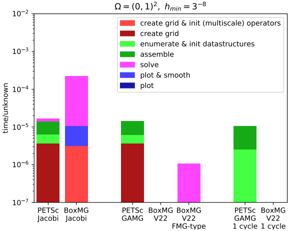
- Convergence speed (optimal)
- Conceptually simple improvements (smoother)

Flaws

- Touch unknowns multiple times
- Assembly time for (multiscale) operators
- Storage footprint of (multiscale) operators

(Additive) Multigrid, BPX, . . .





Likes

- Convergence speed (optimal)
- Conceptually simple improvements (smoother)

Flaws

- Touch unknowns multiple times
- Assembly time for (multiscale) operators
- Storage footprint of (multiscale) operators

Suffers from low AI and high setup cost (assembly)

The root of all evil: We communicate.

Communication-avoiding (to me) is . . .

1. reduction of data volume
2. elimination of (meta) data generation
3. reduction of data exchange frequency
4. homogenisation of data access
5. data access hiding
6. localisation of data transfer

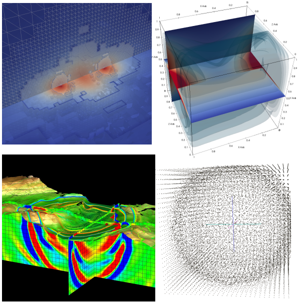
Applies to node-to-node and CPU-to-memory.

This talk

This talk: A tour de force through some communication avoiding techniques tailored to my demonstrators(*).

(*) With the potential to pay off in your projects, too.

Outline



Motivation

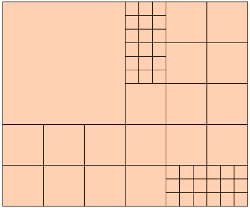
Demonstrators

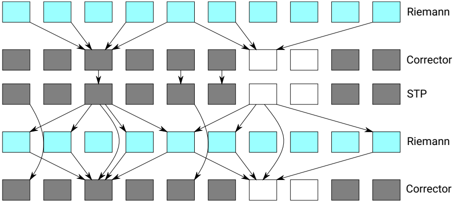
Communication-avoiding techniques

- Avoid assembling data
- Give up on task-traversal association
- Prioritise communication tasks
- Predict synchronisation
- Drop the IEEE standard

Wrap-up

HPC's lingua franca: Task graphs





ADER-DG

- ADER-DG describes a task pattern
- The mesh instantiates the task graph from this pattern
- Task graph conceptually simple
- Algorithm plus mesh plus graph ⇒ one item redundant

PIC and MG

- Conceptually related

Task processing

- Grid traversal = (partial) sweep over task graph

A plea against task graphs

Task assembly-free: Grid traversal spawns tasks, i.e. we use task parallelism without setting up a the task graph.

- Task pattern and instantiate rules exist already anyway
- Task assembly-free
- Mesh traversal acts as “scheduler”
- Avoid expensive assembly (graph changes each and every time step) (*)
- Eliminate meta data footprint (*)
- Best case: Throw away matrix data structures and matrix assembly, too (more on this later)

(*) I might be horribly wrong for static setups/meshes and more complicated algorithms

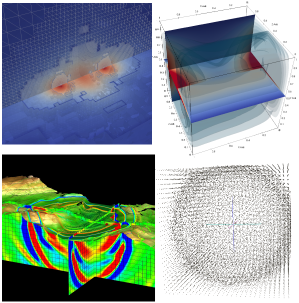
Why I am wrong—a double-edged sword

Task dependencies might require multiple grid sweeps

- Multiple data reads (no single-touch)
- Strong synchronisation between sweeps
- Temporary data
- Tiny tasks

Technique 1: Programming without (graph) assembly.

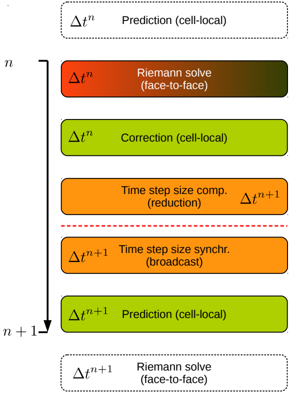
Outline



- Motivation
- Demonstrators
- Communication-avoiding techniques
  - Avoid assembling data
  - Give up on task-traversal association
  - Prioritise communication tasks
  - Predict synchronisation
  - Drop the IEEE standard
- Wrap-up

Single touch implementation: A piece of data is read/written from main memory only once per iteration/time step.

Shifted tasking in ADER-DG


Δt^n Prediction (cell-local)
n
Δt^n Riemann solve (face-to-face)
Δt^n Correction (cell-local)
Time step size comp. (reduction) Δt^{n+1}
Δt^{n+1} Time step size synchr. (broadcast)
Δt^{n+1} Prediction (cell-local)
$n+1$
Δt^{n+1} Riemann solve (face-to-face)

For the time being: assume Δt is known and there's no need to reduce it

Shifted algorithm:

- First read of face: trigger Riemann
- Enter cell: correct solution
  (all $2d$ faces are read already)
- Trigger subsequent STP immediately

Observations:

- Shifted execution model with two activities
  (Riemann and one cell modification)

⇒ Task-fusion
(incl. elimination of temporary memory)

- One time step per sweep

⇒ Amortised single-touch semantics

- Parallel traversal intermixes activities

⇒ Memory access homogenisation

Straightforward application to mat-vecs

1. Partial mat-vec computation: run over grid and contribute per-entity contribution

$$r = Ax$$

2. Apply action of r in next grid traversal

$$x \leftarrow x + \omega diag^{-1}(A)r$$

3. Immediately trigger follow-up computation

$\Rightarrow$ What happens in multiscale systems?

Pipelining in additive multigrid

```
function TDADD($\ell$)                                  ▷ Shift evaluation by half a grid sweep
    $sc_\ell \leftarrow sc_\ell + Psc_{\ell-1}$         ▷ Add coarse grid correction to $sc_\ell$ which
                                                        ▷ so far, holds update resulting from a Jacobi smoothing step.
    $u_\ell \leftarrow u_\ell + sc_\ell + sf_\ell$      ▷ Update $u$ with update from
                                                        ▷ previous line plus all updates done on finer grids.
    $\hat{u} \leftarrow u_\ell - Pu_{\ell-1}$           ▷ Determine new hierarchical surplus.
    if $\ell < \ell_{max}$ then
        TDADD($\ell + 1$)                               ▷ Go to next finer level.
    end if
    $r_\ell \leftarrow b_\ell - H_\ell u_\ell$          ▷ Determine residual and
    $\hat{r}_\ell \leftarrow b_\ell - H_\ell \hat{u}_\ell$   ▷ hierarchical residual.
    $sc_\ell \leftarrow \omega_\ell S(r_\ell)$          ▷ Bookmark update due to a Jacobi
                                                        ▷ smoothing step for next traversal
    if $\ell > \ell_{min}$ then
        $b_{\ell-1} \leftarrow R\hat{r}_\ell$           ▷ Determine right-hand side
                                                        ▷ for multigrid correction.
        $sf_{\ell-1} \leftarrow I(sf_\ell + sc_\ell)$   ▷ Inform other grids about update
                                                        ▷ that is due in next traversal
                                                        ▷ Update propagation aligns with traversal data flow
    end if
end function
```

Shifting summary

Lessons learned

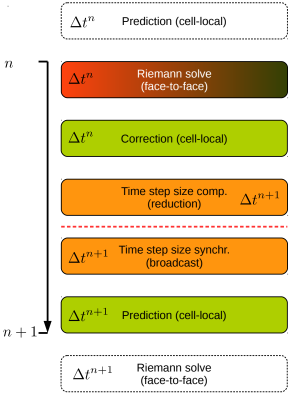
- Suffered from tight association of grid traversals with algorithmic steps
- Don't eliminate temporary data on paper immediately (residual, unknown updates) but use them for pipelining
- Desynchronise grid entities (some already are ahead in the computation) ⇒ anarchic mindset
- Not too difficult to prove correctness

Open questions

- Increased memory footprint
- Stability(*)
- Causal dependencies

(*) Cmp. work by Vanroose et al. Likely not an issue here as only one step of pipelining.

Be optimistic and anarchic



Δt^n Prediction (cell-local)

n

Δt^n Riemann solve (face-to-face)

Δt^n Correction (cell-local)

Time step size comp. (reduction) Δt^{n+1}

Δt^{n+1} Time step size synchr. (broadcast)

Δt^{n+1} Prediction (cell-local)

$n+1$

Δt^{n+1} Riemann solve (face-to-face)

Move orange sync down

- Intermix compute phases over different grid entities

  (data access homogenisation)
- Bring STP calculations forward
- Fuse correction with STP tasks
- Be optimistic (and eliminate synchronisation):
  - Work with estimate $\Delta t_{est}^{(n+1)}$
  - Determine $\Delta t_{adm}^{(n+1)}$ on-the-fly
  - If admissible $\Delta t_{adm}^{(n+1)} > \Delta t_{est}^{(n+1)}$ use $\Delta t_{est}^{(n+1)} \leftarrow 0.5(\Delta t_{adm}^{(n+1)} + \Delta t_{est}^{(n+1)})$
  - If $\Delta t_{adm}^{(n+1)} < \Delta t_{est}^{(n+1)}$ reset $\Delta t_{est}^{(n+1)} \leftarrow 0.9\ \Delta t_{adm}^{(n+1)}$

$\Rightarrow$ AMR, time step decreases might make approach fall back to two-sweep paradigm

$\Rightarrow$ Does seem to happen negligible often

- PIC: Same time stepping mindset
- MG: Postpone analysis of termination criterion

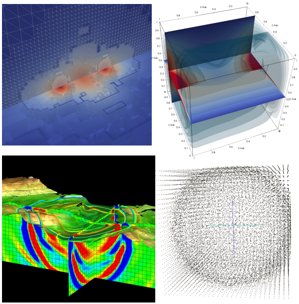
Some metrics for ADER-DG

	1 core			12 cores			24 cores		
p	BW	Volume	Time	BW	Volume	Time	BW	Volume	Time
3	1,357.44	30.59	1.36	2,885.94	35.24	0.75	4,639.00	73.32	0.71
5	1,155.42	101.64	3.91	4,158.00	117.06	1.25	6,377.94	223.06	1.04
7	806.87	215.91	14.75	5,695.60	285.40	2.24	8,540.10	520.33	1.83
9	483.04	487.98	29.20	20,894.15	4,376.39	4.66	30,938.36	4,716.02	3.76
3	1,233.97	24.05	1.14	3,645.78	31.58	0.39	5,481.20	71.92	0.39
5	861.10	80.49	4.18	5,931.70	110.40	0.68	8,403.62	211.44	0.57
7	625.40	176.84	10.71	6,877.66	350.95	1.98	9,003.53	621.64	1.50
9	429.35	434.20	25.96	17,525.00	4,619.24	4.80	27,297.87	5,280.57	4.32

Performance counters for a 27 × 27 × 27 grid. Upper part: straightforward 3-step implementation. Lower part: fused approach. The bandwidth (BW) is given as MB/s, the volume (Vol.) transferred is given in GB, all timings are time per time step with $[T] = s$.

- the lower the AI the higher the reward
- optimistic assumption here failed in $\ll 1\%$
- for very expensive STPs still burst of memory accesses
  (shift's homogenisation of memory accesses not sufficient)

Outline



- Motivation
- Demonstrators
- Communication-avoiding techniques
  - Avoid assembling data
  - Give up on task-traversal association
  - Prioritise communication tasks
  - Predict synchronisation
  - Drop the IEEE standard
- Wrap-up

Flaws in the task graph-free approach

Task assembly-free: Grid traversal spawns tasks, i.e. we use task parallelism without setting up a the task graph.
⇒ Traversal fires (and forgets) tasks.

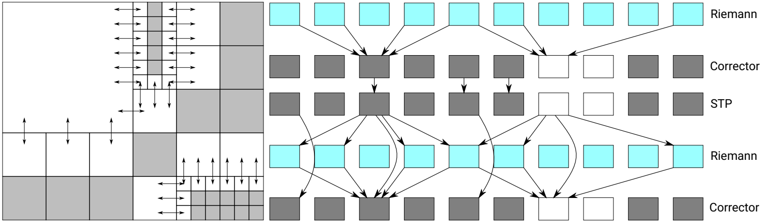
Barely a new idea ignoring some showstoppers

(on parallel computers)

In an ideal world, we would like to run through the whole grid with one large `parallel for` issuing face and cell operations, but

- cells along MPI boundaries have to be processed in order
  (MPI messages may not overtake)
- all cores might run Riemann (or any particular task type) concurrently
  (see experimental evidence for $p = 9$)
- dynamic AMR changes mesh between corrector and subsequent STP
  (memory allocation and initialisation/FV reruns synchronise code)
- AMR requires particular inter-grid transfer operator orders
  (multi-resolution data consistency)

Enclave tasking—inspired by Ghattas et al.



- Mark all cells along MPI boundary and resolution transitions ⇒ skeleton grid
  (those are involved in MPI and might refine/coarsen; this is an optimistic assumption)

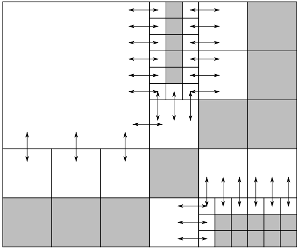
⇒ reordering challenging

- Give up on idea to run Riemann solves parallel
  (at least on all of the cores)

⇒ bandwidth-bound
  (cheap and skeleton operations done immediately and close-to sequentially)

- Make remaining cells (enclaves) yield the scaling
  (job stealing brings in idling cores)

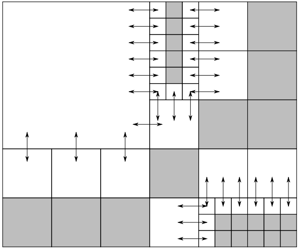
Prioritise tasks



Primary sweep:

- Per face
  - wait for adjacent STPs to finish (cmp. shifting)
  - Riemann solves
  - restrict image along resolution transitions
- Per cell
  - Corrector (bottom-up traversal)
  - Dynamic adaptivity, limiter reruns, . . .
  - determine skeletons on-the-fly
- Spawn STP
  - on skeleton: high priority
  - on enclaves: low priority (background)

Prioritise tasks



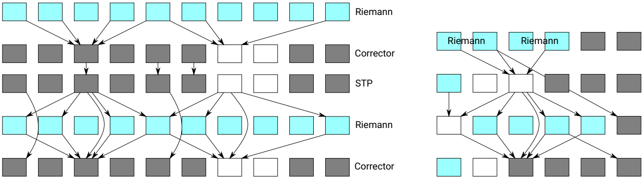
Primary sweep:

- Per face
  - wait for adjacent STPs to finish (cmp. shifting)
  - Riemann solves
  - restrict image along resolution transitions
- Per cell
  - Corrector (bottom-up traversal)
  - Dynamic adaptivity, limiter reruns, . . .
  - determine skeletons on-the-fly
- Spawn STP
  - on skeleton: high priority
  - on enclaves: low priority (background)

Secondary (partial) sweep:

- Ignore enclaves (partial traversal)
- Per skeleton cell: wait for STP
- Trigger MPI exchange
- Interpolate along AMR transitions for next Riemann solve

Enclave implications

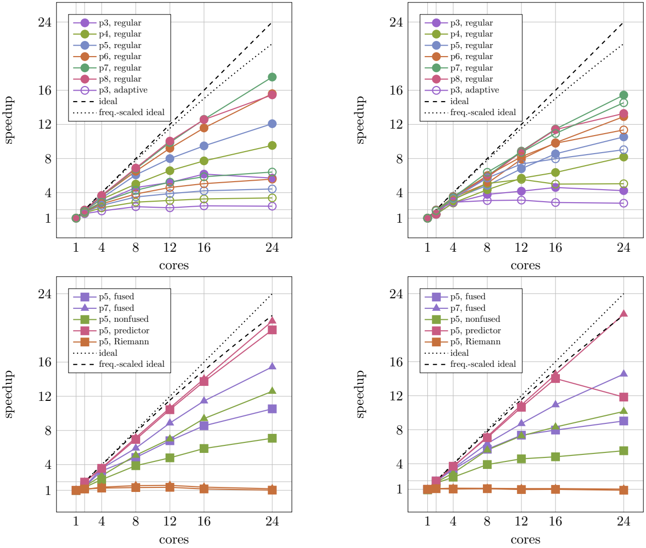


- Low intensity tasks constantly trickle through system
  (homogenise arithmetic load/avoid memory access bursts)
- Coarsening & refinement accompanied by background/enclave STPs
  (avoid memory allocation and initialisation not throttling everybody else)
- STPs yielding MPI messages ran before majority of (enclave) tasks
  (more time to overlap messaging and processing)

Implementation remarks:

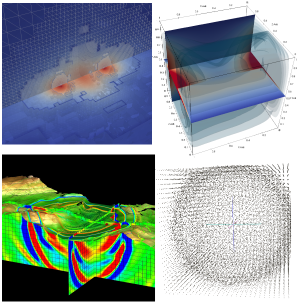
- Skeleton traversal has highest priority ⇒ tweak TBB runtime
- Fuse multiple enclave tasks to reduce overhead ⇒ tweak TBB runtime

`parallel for` vs. enclaves



Broadwell; top row: regular grid `parallel for` vs. tasks; bottom row: regular (left) vs. adaptive (right)

Outline



Motivation

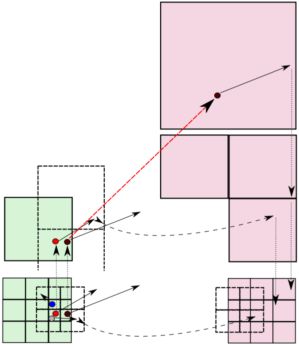
Demonstrators

Communication-avoiding techniques

- Avoid assembling data
- Give up on task-traversal association
- Prioritise communication tasks
- Predict synchronisation
- Drop the IEEE standard

Wrap-up

The particle code



Single touch through shifts

- Bookmark position update
  (do not apply)
- Apply position update as preamble to next step
  (first touch)
- Update grid-particle association in preamble, too

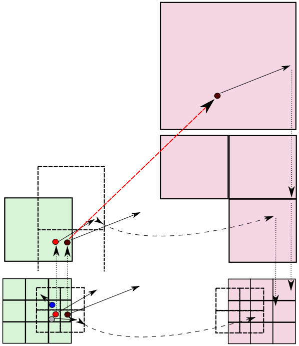
⇒ simple iff particles travel at most one cell per time step

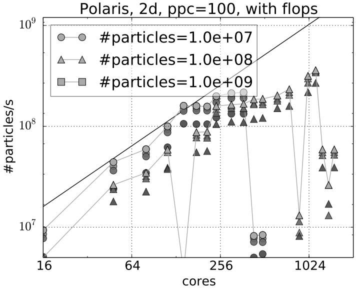
Data maintenance

- Use cascade of grids (octree)
- Particles on finest level for computation
- Otherwise on level such that they drift at most one level

⇒ lift-n-drop mechanism

The particle code



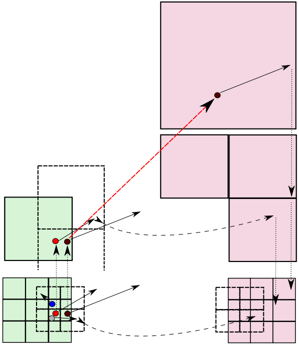


Horizontal data exchange

- ▶ Along multiscale domain boundaries
  (non-blocking MPI hiding behind computation)
- ▶ Very fast particles are lifted aggressively
- ▶ Non-neighbour MPI ranks involved (global sort)

⇒ Latency-sensitive

Predict data flow



Vertical data flow:

- Max velocity $v_{max}(c)$ per cell $c \in \mathcal{T}$
- Extrapolate velocity updates (fine grid)
- Analysed tree grammar

$$v_{max}(c) = \begin{cases} max_{p \in c}|v(p)| & \text{for leaves} \\ max_{c'}\ v_{max}(c') & \text{otherwise} \\ & \forall\ c' \sqsubseteq_{child\ of}\ c \end{cases}$$

- Next traversal:
  - Particle p drops into domain

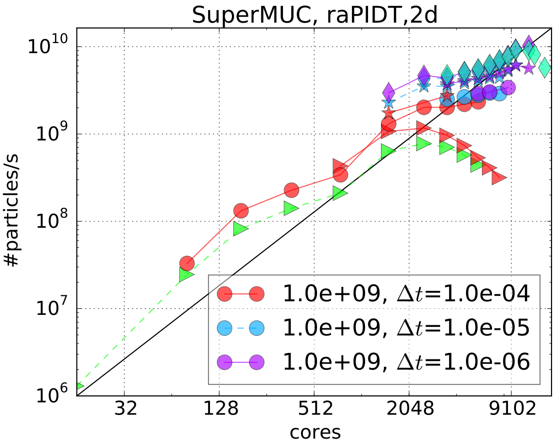
    $$v_{max} \leftarrow max(v_{max}, |v(p)|)$$

  - $v_{max} \leq h/\Delta t$: no tunnelling possible and likely not to happen either

  $\Rightarrow$ skip reduction for these rank pairs

  - Security factor for extrapolation

reduction-avoiding Particle-In-Tree

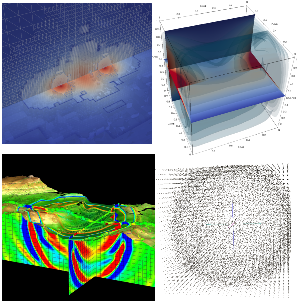


- Triangle: $2.0 \cdot 10^8$ particles
- Diamond: $4.0 \cdot 10^9$ particles
- Star: $1.0 \cdot 10^{10}$ particles
- Colours encode time step sizes

Generic patterns

- Communication graph replaces collective
- Sparsify a priori
  (sparsity of communication graph anticipates particle positions and velocities)
- Similar pattern used to predict dynamic mesh refinement, wave fronts, time step size evolution

Outline



Motivation

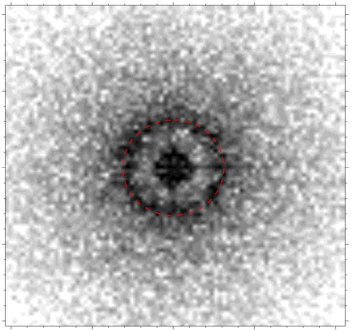
Demonstrators

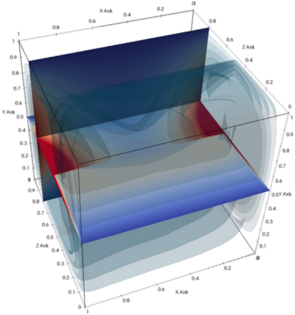
Communication-avoiding techniques

- Avoid assembling data
- Give up on task-traversal association
- Prioritise communication tasks
- Predict synchronisation
- Drop the IEEE standard

Wrap-up

Massive memory

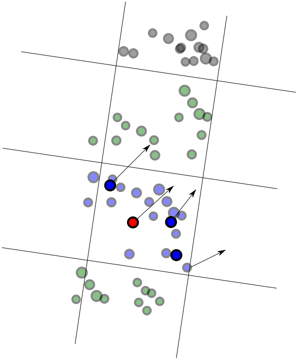




- More particles more accurate physics
- Single-touch nice, but then homogeneously memory bound

- Pure geometric operators unstable
- Matrix-free not really an option

Information density



Particle sets

- Particles per cell carry similar values (often)
- Compute average attribute $\widetilde{A}(c)$ per cell
- Encode hierarchical $\hat{A}(p) = A(p) - \widetilde{A}(c)$ per cell
- Compute on-the-fly how many bytes per attribute are required such that

$$\forall p \in c : |f_{bpa}(\hat{A}(p)) - \hat{A}(p)| \leq \epsilon$$

MG operators

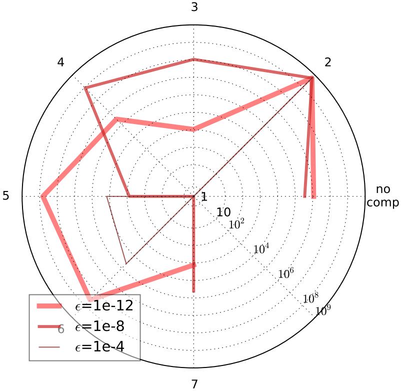
- Operators often similar to Poisson/d-linear
- Compute difference to these stencils
- Encode hierarchical

Both approaches: convert back prior to next usage

Reproducible science

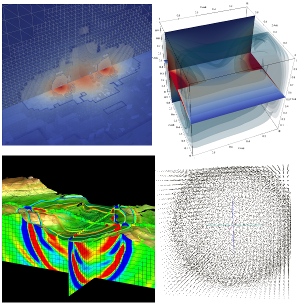
```
void peano::heap::decompose( double value, char exponent[8], long int mantissa[8], double error[8] ) {
   int shiftExponent = 6;
   const long int sign = value < 0.0 ? -1 : 1;
   if (sign<0) {
     value = -value;
   }
   int            integerExponent;
   const double   significand = std::frexp(value , &integerExponent);
   for (int i=0; i<8; i++) {
     const double shiftMantissa = std::pow( 2.0,shiftExponent );
     exponent[i]  = static_cast<char>( integerExponent-shiftExponent );
     mantissa[i]  = static_cast<long int>
       ( std::round(significand*shiftMantissa) );
     error[i] = std::abs( std::ldexp(mantissa[i],exponent[i]) - value );
     std::bitset<64>*  mantissaAsBitset =
       reinterpret_cast<std::bitset<64>*>( &(mantissa[i]) );
     if (sign<0) {
       mantissaAsBitset->flip( (i+1)*8-1 );
     }
     shiftExponent+=8;
   }
}
```

Forget IEEE



- Reduction of memory footprint
- “Almost” matrix-free for multigrid
- Generic observation: double values do often not carry enough significant bits compared to their neighbours/averages/recomputed data
- On-the-fly analysis: only compress where error is not too big

Outline



- Motivation
- Demonstrators
- Communication-avoiding techniques
  - Avoid assembling data
  - Give up on task-traversal association
  - Prioritise communication tasks
  - Predict synchronisation
  - Drop the IEEE standard
- Wrap-up

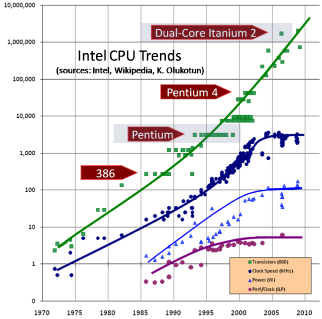
My notion of “communication-avoiding”

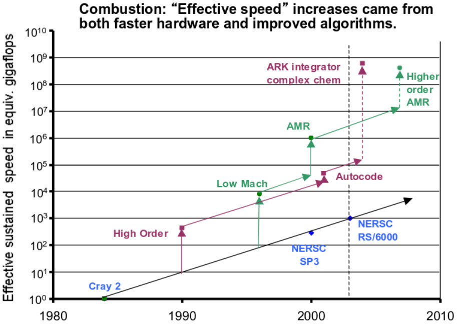
1. reduction of data volume (IEEE)
2. elimination of (meta) data generation (assembly)
3. reduction of data exchange frequency (shifts, fusion, prediction and batching)
4. homogenisation of data access (scheduling, shifts and enclaves)
5. data access hiding (enclaves)
6. localisation of data transfer (prediction)

- Generalised interpretation of “communication-avoiding”
- Applies to both message exchange and on-chip data access
- Might still be incomplete . . .
- Requires us to rethink our implementations
- Benefits from generic implementation techniques
- Hunt for these techniques has just started

Message: It is more than posting sends and receives early. It is more than avoiding replicated messages. It is more than eliminating temporary data.

Context




Combustion: “Effective speed” increases came from both faster hardware and improved algorithms.

H. Sutter: The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software, 2005 (left)

D. Keyes: SCaLeS Report, Vol. 2, 2004 (right)

- Some patterns have made/kicked off multifaceted impact:
  - Think multiresolution
  -
- Time might be right for similar impact of communication avoiding techniques

Famous last words

It is all open source (`www.peano-framework.org` or `www.exahype.eu`).

Support & Grants

- Peano (www.peano-framework.org) has been developed for more than 10 years under various grants made by Technische Universität München (TUM), KAUST, German Research Foundation (DFG), Durham University, and many more.
- ExaHyPE (www.exahype.eu) has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 671698.

  ExaHyPE
  An Exascale Hyperbolic PDE Engine

  European Commission
  Horizon 2020
  European Union funding
  for Research & Innovation

- The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer SuperMUC at Leibniz Supercomputing Centre (LRZ, www.lrz.de).

  lrz Leibniz Supercomputing Centre
  of the Bavarian Academy of Sciences and Humanities

- Experiments were made possible through Durham's supercomputer Hamilton.

  Durham
  University

- We appreciate early-adopters access to Intel Xeon Phi 7200 (codenamed KNL) family processors through the RSC Group.

  RSC

Support & Grants

- Peano (www.peano-framework.org) has been developed for more than 10 years under various grants made by Technische Universität München (TUM), KAUST, German Research Foundation (DFG), Durham University, and many more.
- ExaHyPE (www.exahype.eu) has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 671698.

ExaHyPE
An Exascale Hyperbolic PDE Engine

European
Commission

Horizon 2020
European Union funding
for Research & Innovation

- The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer SuperMUC at Leibniz Supercomputing Centre (LRZ, www.lrz.de).

lrz
Leibniz Supercomputing Centre
of the Bavarian Academy of Sciences and Humanities

- Experiments were made possible through Durham's supercomputer Hamilton.

Durham
University

- We appreciate early-adopters access to Intel Xeon Phi 7200 (codenamed KNL) family processors through the RSC Group.

RSC

Some links

- `https://miscada.phyip3.dur.ac.uk`

MSc SCIENTIFIC COMPUTING AND DATA ANALYSIS

- `www.peano-framework.org` and `www.exahype.eu`

ExaHyPE
An Exascale Hyperbolic PDE Engine

- `dur.ac.uk/tobias.weinzierl`