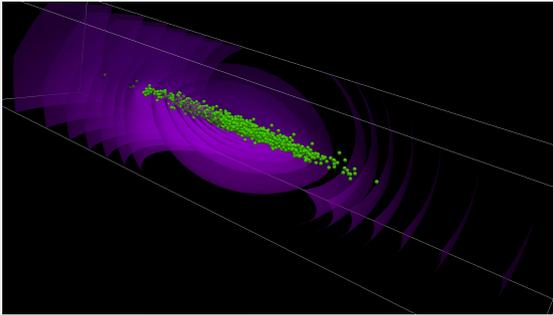


Accelerator Beam
Dynamics on
Multicore and GPU
and MIC Systems

James Amundson
and
Qiming Lu
Fermilab

Synergia



Synergia: A comprehensive
accelerator beam dynamics package

<http://web.fnal.gov/sites/synergia/SitePages/Synergia%20Home.aspx>



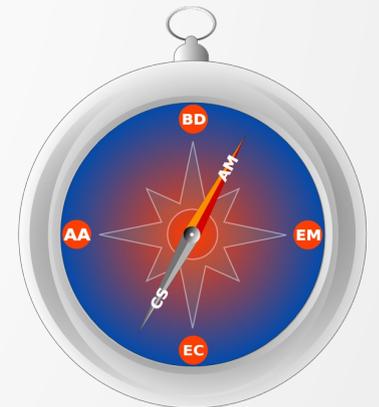
Accelerator Simulation Group

James Amundson, Paul Lebrun, Qiming Lu, Alex Macridin, Leo Michelotti, Chong Shik Park, (Panagiotis Spentzouris) and Eric Stern

The ComPASS Project

High Performance Computing for Accelerator Design
and Optimization

<https://sharepoint.fnal.gov/sites/compass/SitePages/Home.aspx>



Funded by DOE SciDAC

Topics

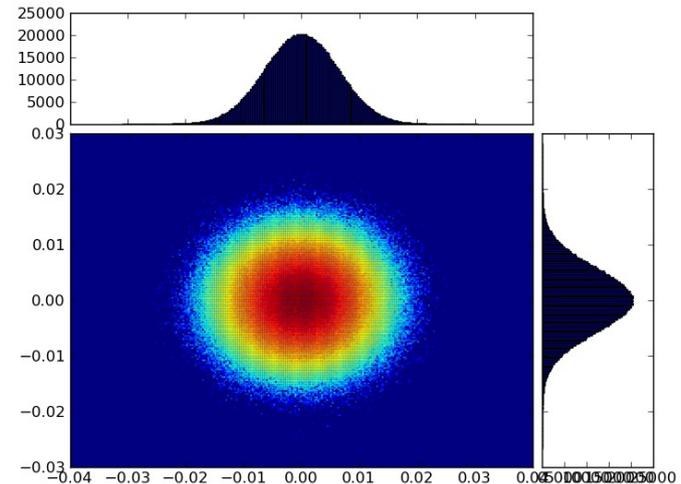
- Accelerator beam dynamics
- Parallel scaling in Synergia
- Why we need to move on
- A GPU implementation
- Early MIC results
- Toward a next-generation Synergia

Computational Beam Dynamics

- Existing and planned accelerators
 - 1,000s of elements
 - 10s of *types* of elements
 - 1,000s to 1,000,000s of revolutions
 - 1-1000s of bunches of $O(10^{12})$ particles



- 50-1000 steps/revolution
- Internal and external fields
 - External field calculations trivially parallelizable
 - Internal field calculations require PIC
- Minimal bunch/field structure

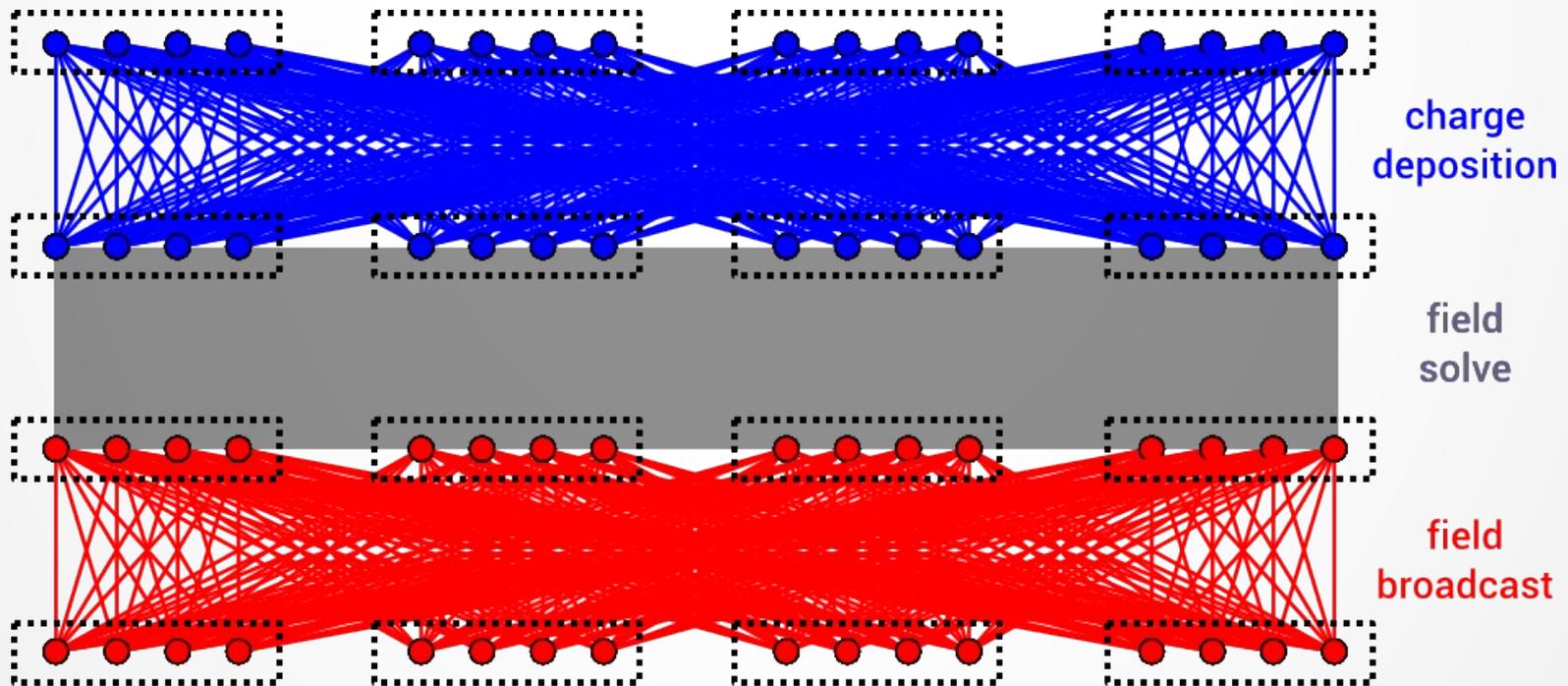


Parallel Scaling in Beam Dynamics

- Challenge: beam dynamics simulations are big problems require many small solves
 - Typically $64^3 - 128^3$ grids ($2e5 - 2e6$ degrees of freedom)
 - Need to do many time steps ($1e5$ to $1e8$)
- Typical pure-PIC scaling applies to scaling with respect to grid size
 - Including decomposing particles by grid location
 - In beam dynamics, external fields can cause particles to move over many grid cells in a single step
 - Communication required to maintain decomposition and load balance
 - Point-to-point communication
 - Complicated for both programmer and end-user
 - Change in physical parameters can change communication time by x100

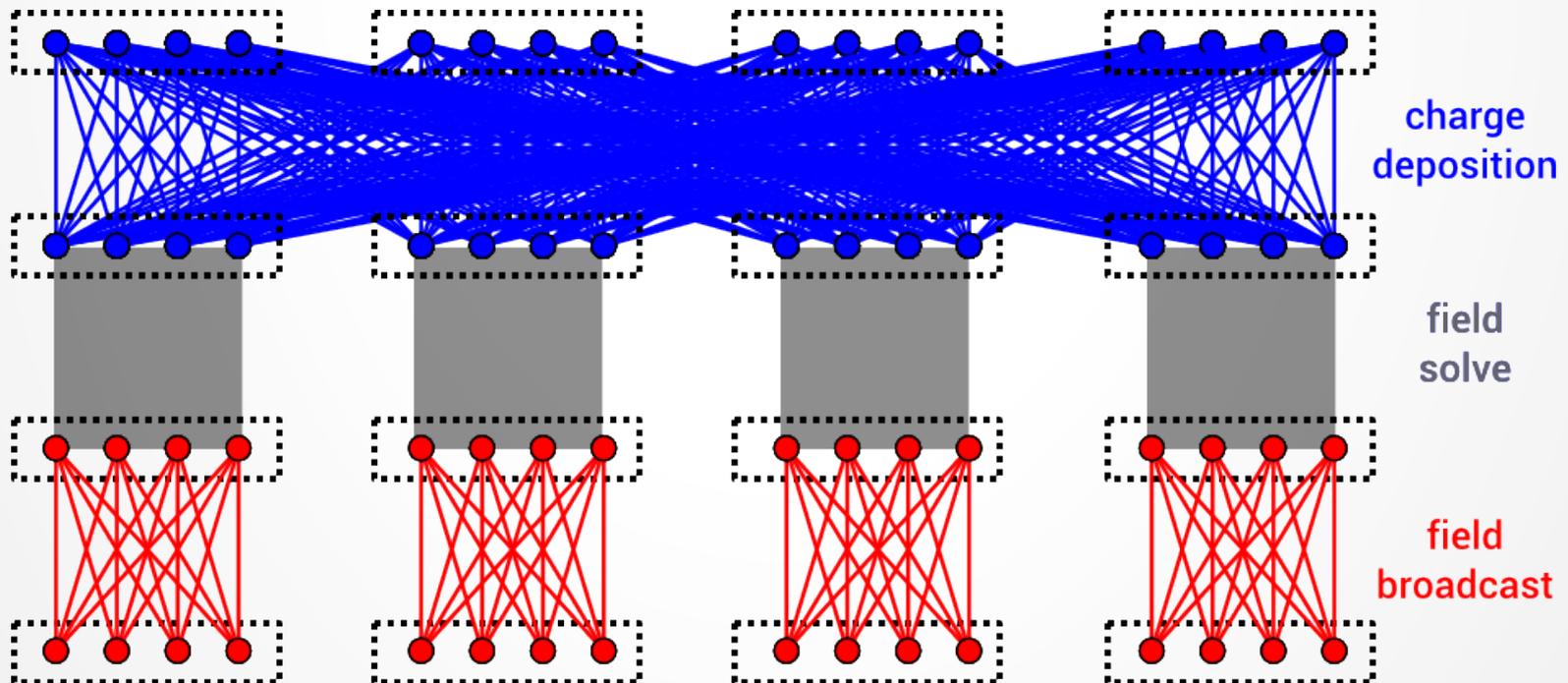
Particle (lack of) decomposition

- First step: eliminate particle decomposition
 - Requires collective communication
 - But not point-to-point
 - Collectives are typically highly optimized
 - Simpler for programmer and end-user



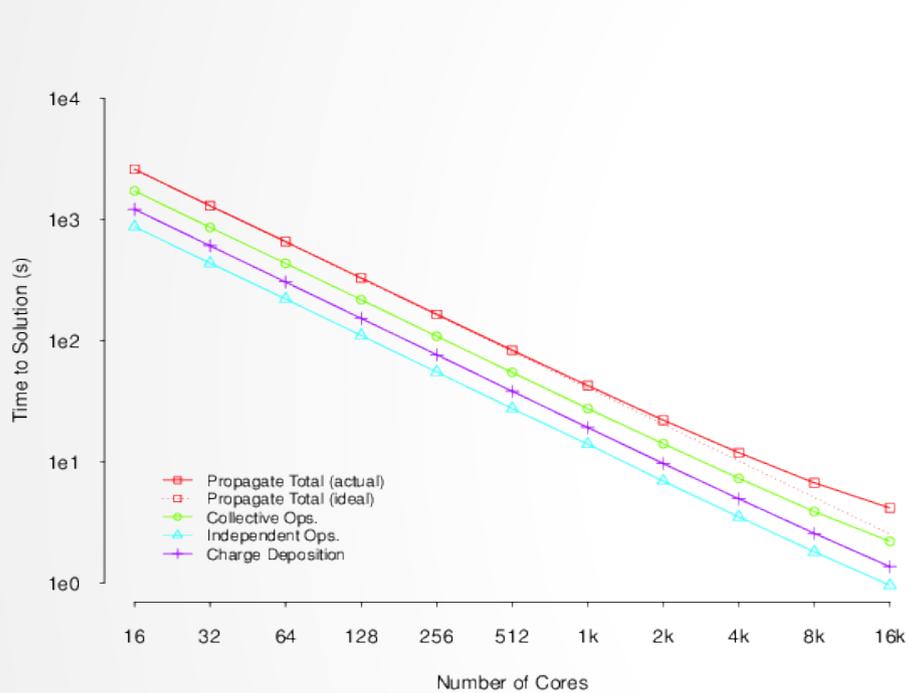
Communication avoidance

- Second (breakthrough!) step: redundant field solves
 - Field solves are a fixed size problem
 - More calculation, less communication
 - Allows scaling in number of particles and/or bunches
 - Can use arbitrary unit size, but one node is usually best

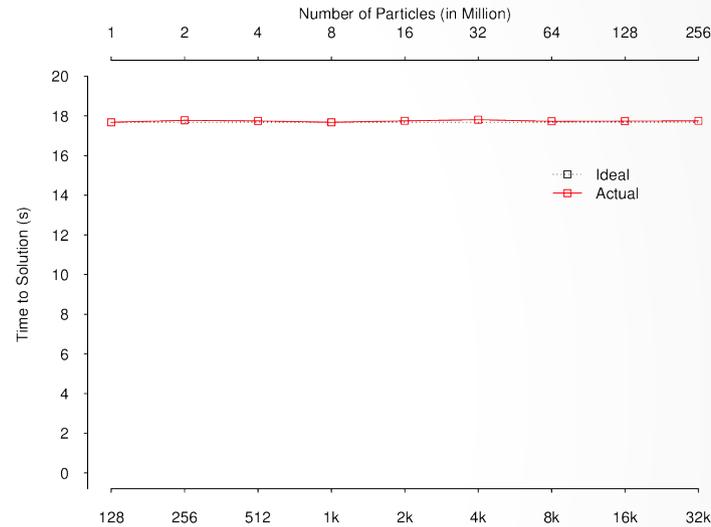


Scaling

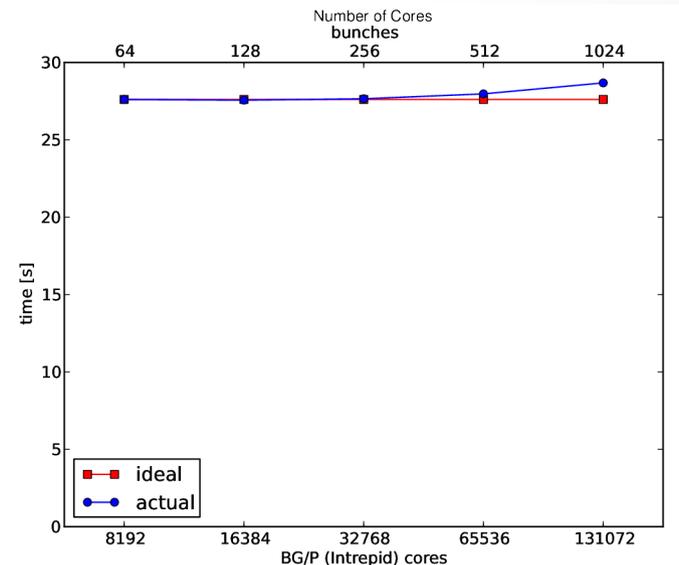
Scaling results on ALCF machines: Mira (BG/Q) and Intrepid (BG/P)



Single-bunch strong scaling from
16 to 16,384 cores
32x32x1024 grid, 105M particles



Weak scaling from
1M to 256M
particles
128 to 32,768 cores



Weak scaling from
64 to 1024 *bunches*
8192 to 131,072
cores
Up to over 10^{10}
particles

Why am I here?

- Synergia runs on a wide variety of platforms



ODROID-U3
(ARM A9)



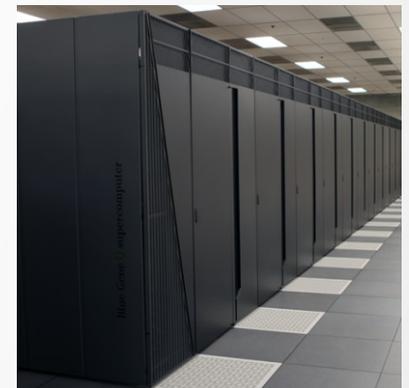
laptops and desktops



Linux clusters



Cray



Blue Gene

Why? continued

- Interesting and useful beam dynamics cover a tremendous range of computing requirements
 - All of the platforms on the previous slide are useful
 - We typically emphasize the extraordinarily large
 - Our specialty
- We target all end users, including those who are not experts in cluster- or supercomputing

Why GPU/MIC/Multicore?

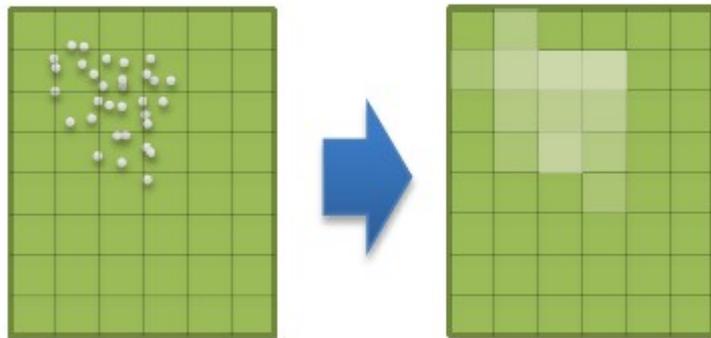
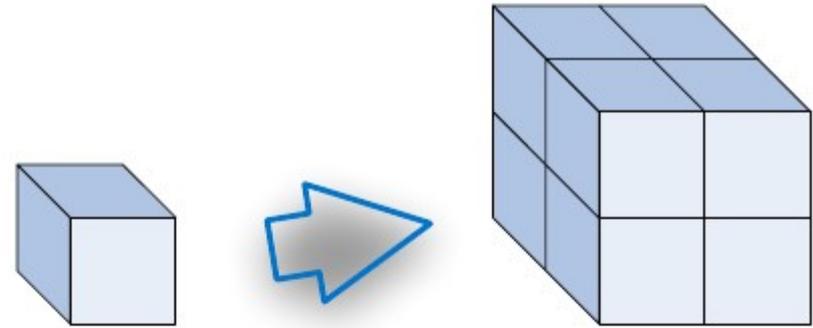
- The future of supercomputing
- The future of desktop computing
- I anticipate a new class of target hardware: single box with a few GPUs and/or MICs
 - Cheaper to obtain and maintain than a Linux cluster
 - Easier to use

Optimizing for GPUs and Multicore

- Shared memory is back!
 - Some things get easier, some harder
- Cache coherency in shared memory systems is the key challenge
- Multi-level parallelism very compatible with our communication avoidance approach

Charge deposition in shared memory

One macro particle contributes up to 8 grid cells in a 3D regular grid



Collaborative updating in shared memory needs proper synchronization or critical region protection

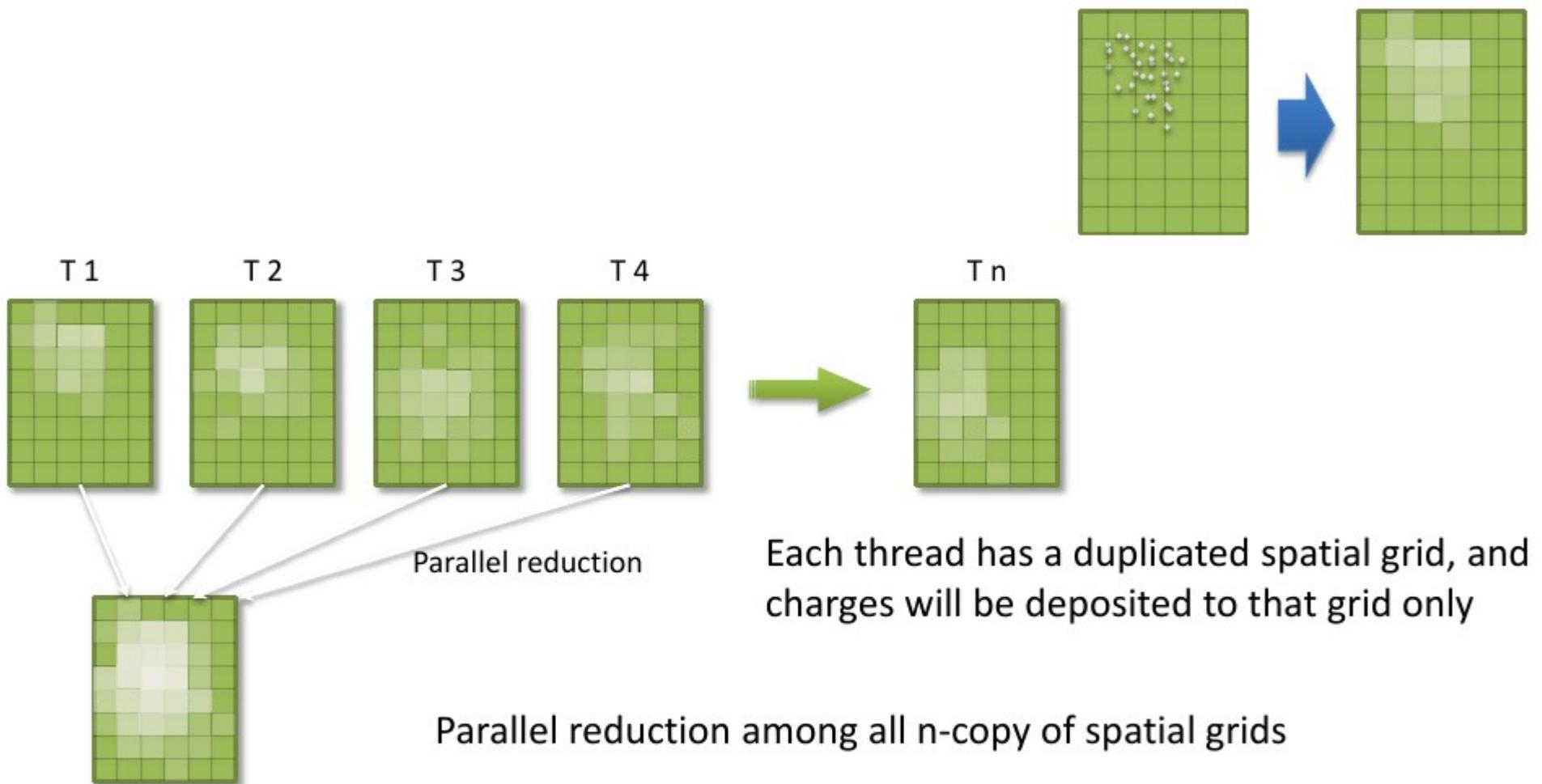
CUDA

- No mutex, no lock, no global sync
- Atomic add – yes, but not for double precision types

OpenMP

- ***#pragma omp critical***
- ***#pragma omp atomic***
- ***Both very slow***

Charge deposition in shared memory – solution 1



CUDA

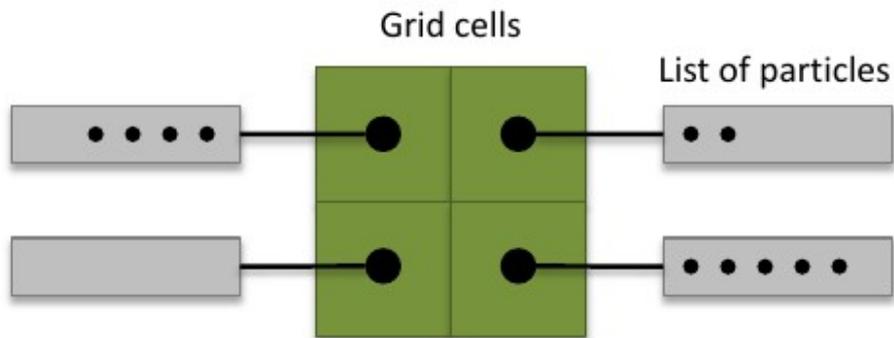
- Concurrency be an issue for GPU
- Memory bottleneck at final reduction

OpenMP

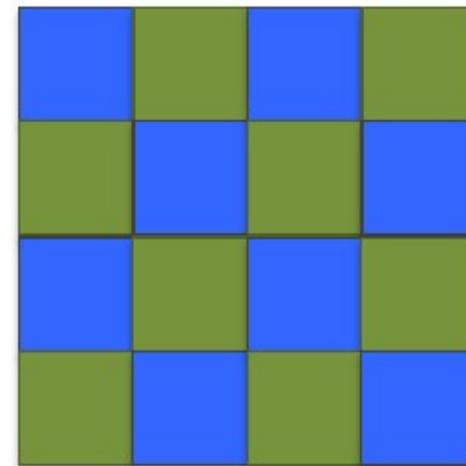
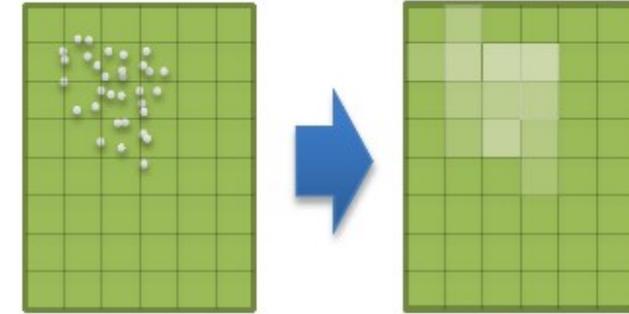
- Works well at 4 or 8 threads
- Scales poorly at higher thread counts

Charge deposition in shared memory – solution 2

Sort particles into their corresponding cells using parallel bucket sort



Deposit based on color-coded cells in an interleaved pattern (red-black)



CUDA

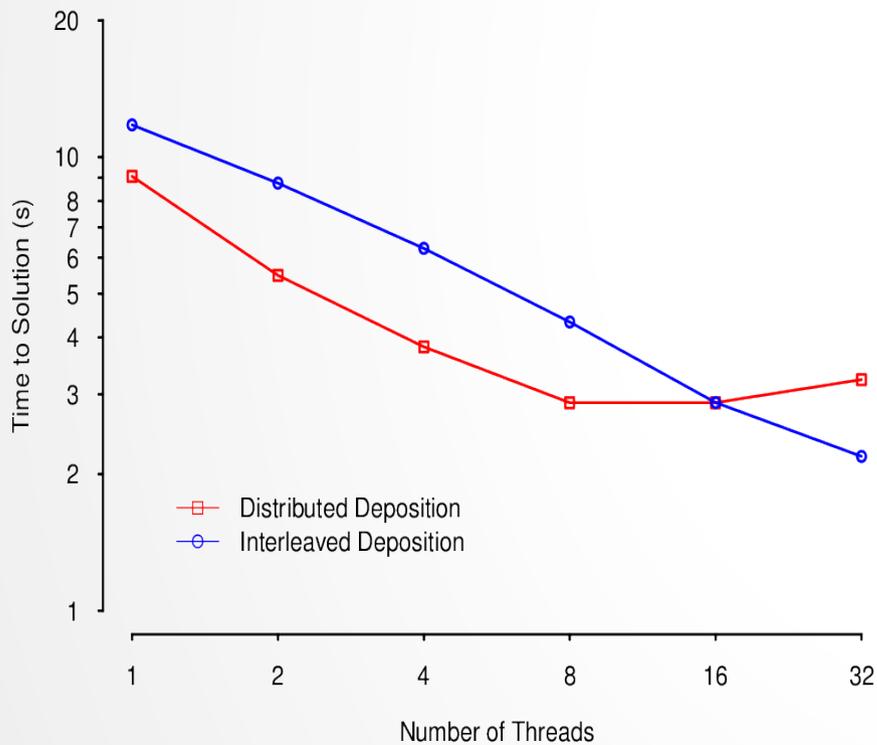
- High thread concurrency
- Good scalability, even the overhead shows reasonable scaling
- No memory bottleneck
- Better data locality at pushing particles

OpenMP

- Non-trivial sorting overhead for low thread counts

GPU and Multicore Results

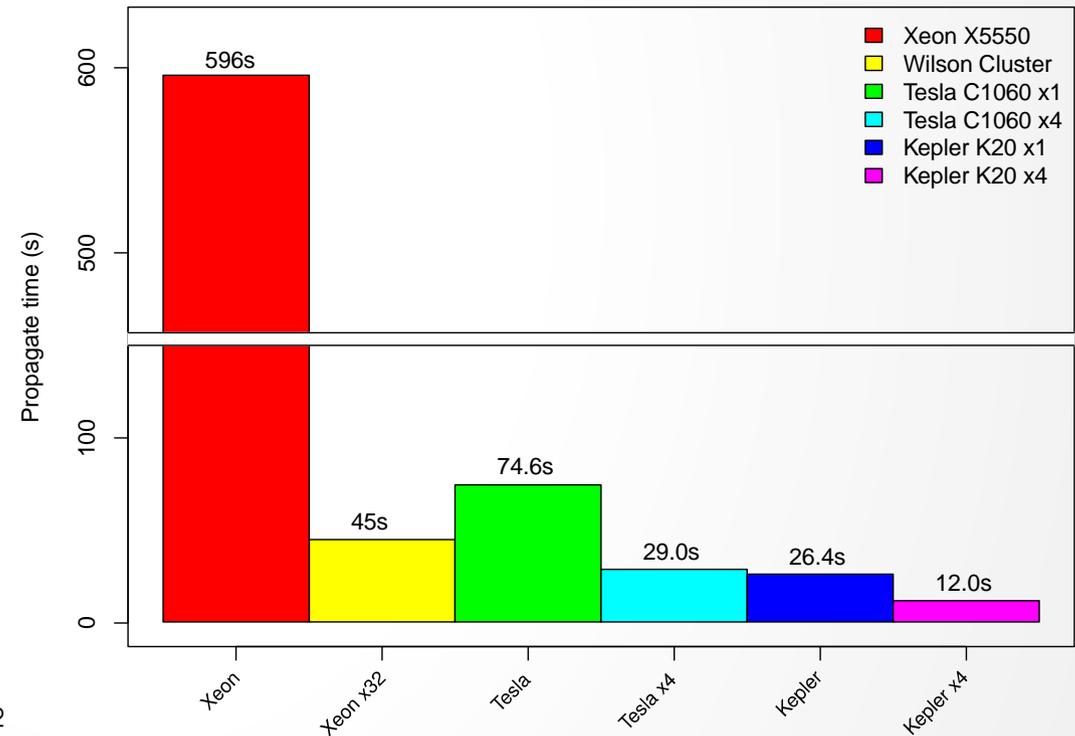
OpenMP results



Just charge deposition

GPU results

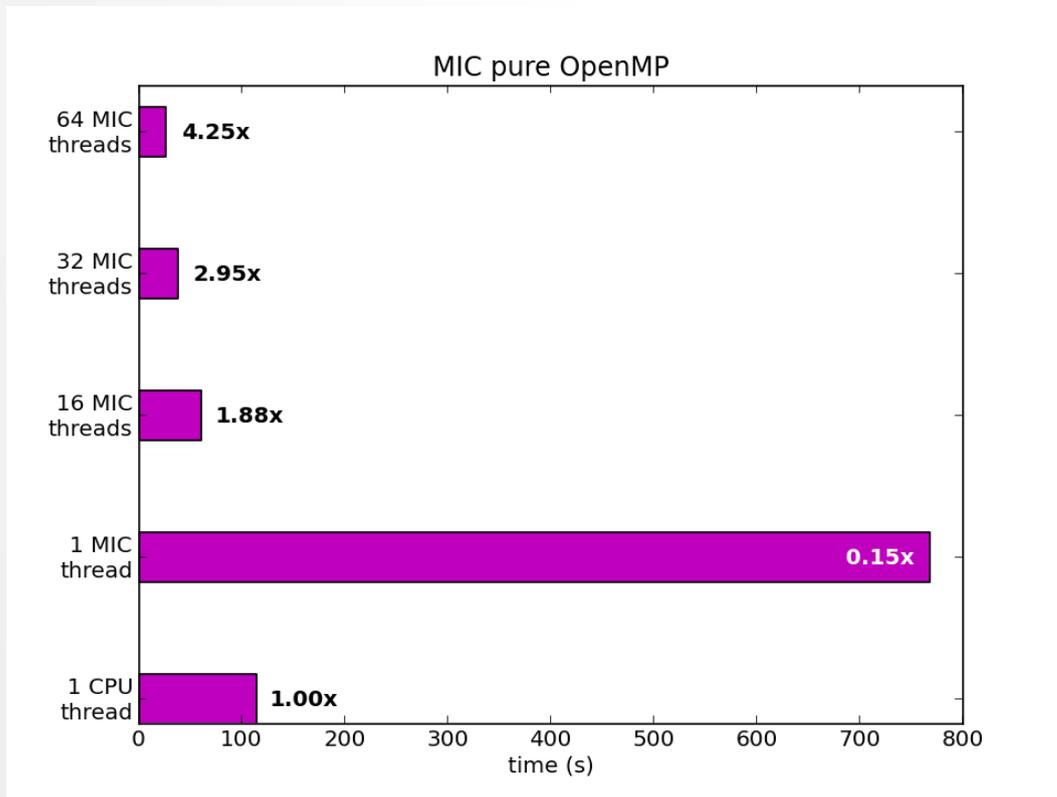
Comparison of CPUs and GPUs



Full (toy) simulation

Intel MIC results

- One selling point for Intel MIC architecture: cores are similar to traditional Intel architecture – major changes may not be necessary...



MIC pure MPI



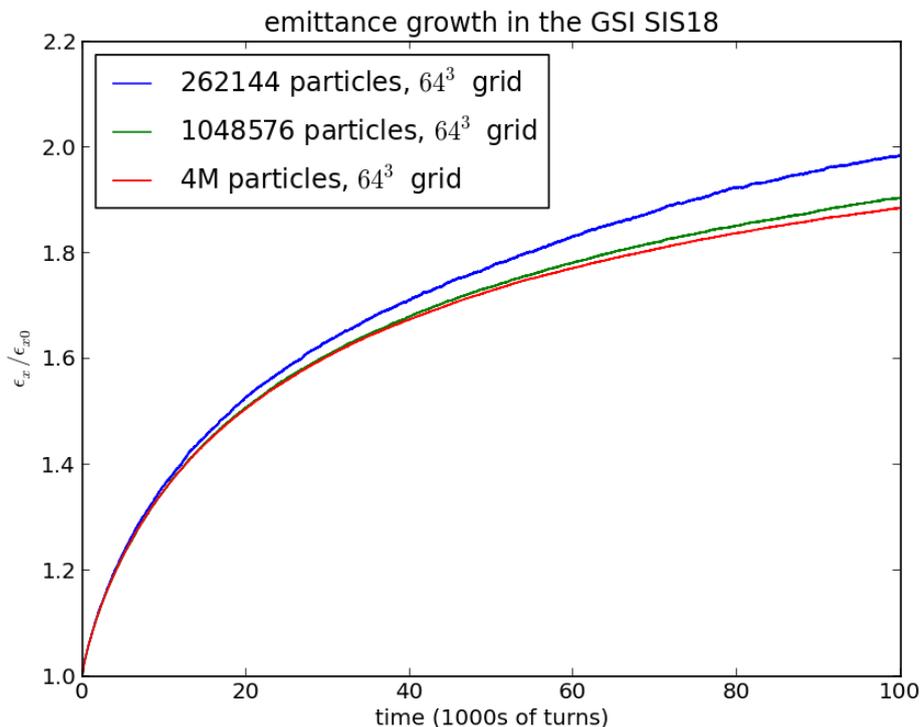
What next?

- Need to take advantage of MIC SIMD
- Early criticism of GPU “speedups”: What if you spent the same effort optimizing your non-GPU code?
- Work toward unified production code
 - Leverage optimization efforts for all architectures

	cluster	supercomputer	MIC	GPU
communication avoidance	X	X	X	X
memory offloading			X	X
SIMD	X	X	X	0

Real-world problem (in progress)

- Study emittance growth over 100,000 revolutions in GSI SIS18 accelerator
 - Effects of statistical noise appear to be important



71 steps/turn

7,100,000 steps

4,194,304 particles

29,779,558,400,000 particle-steps

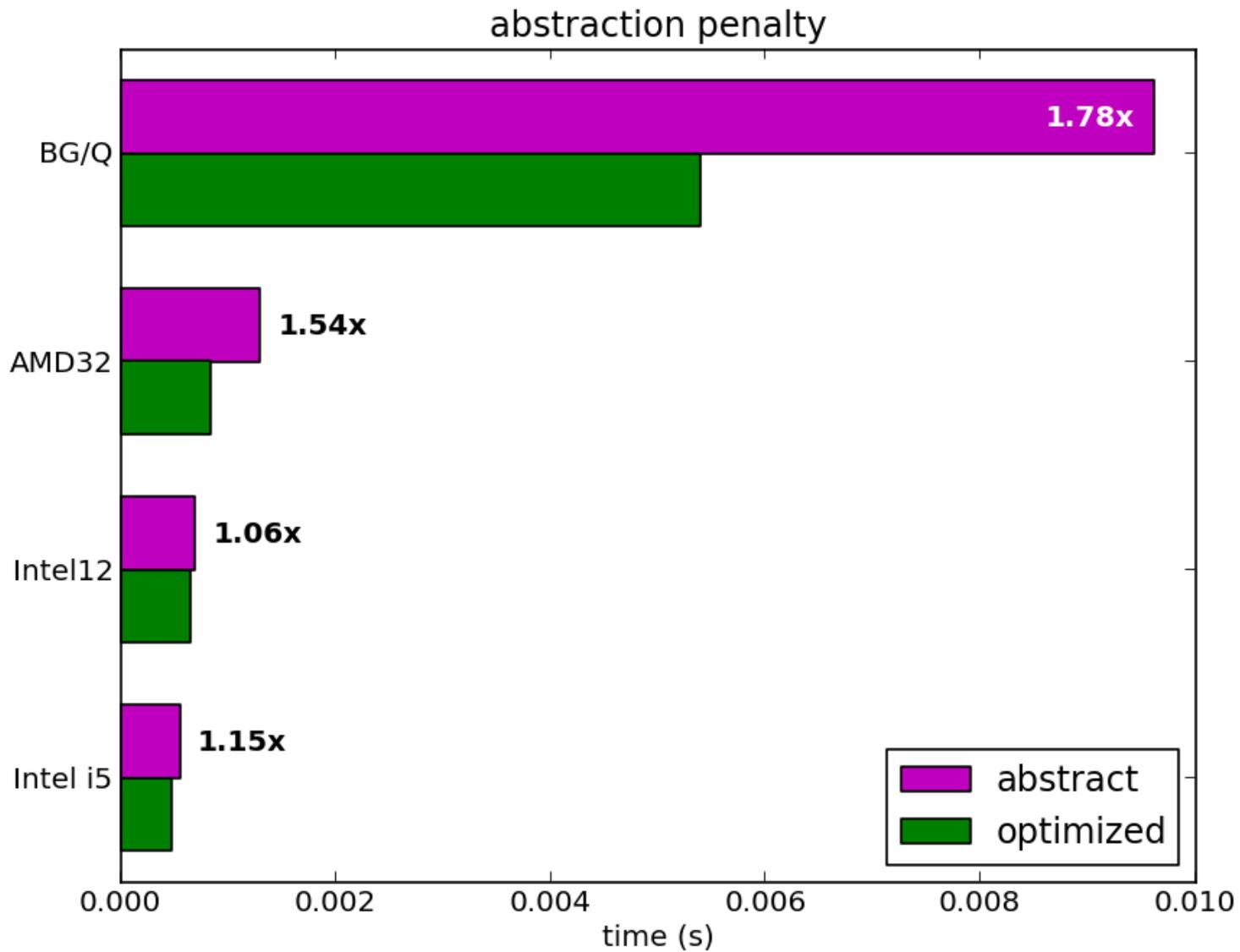
1,238,158,540,800,000 calls to “drift”

Yes, that's over a quadrillion

Optimizing “drift”

- External field calculations (including drift) in Synergia provided by CHEF
 - C++, predates Synergia by over a decade
 - Designed for deep analysis of single-particle physics
 - The genius: the same code propagates particle coordinates and polynomials in particle coordinates
 - Non-linear map analysis
 - Synergia has to convert each of its particles to a CHEF particle (and back) each half step
 - 60 trillion conversions each way
 - This overhead is our “abstraction penalty”

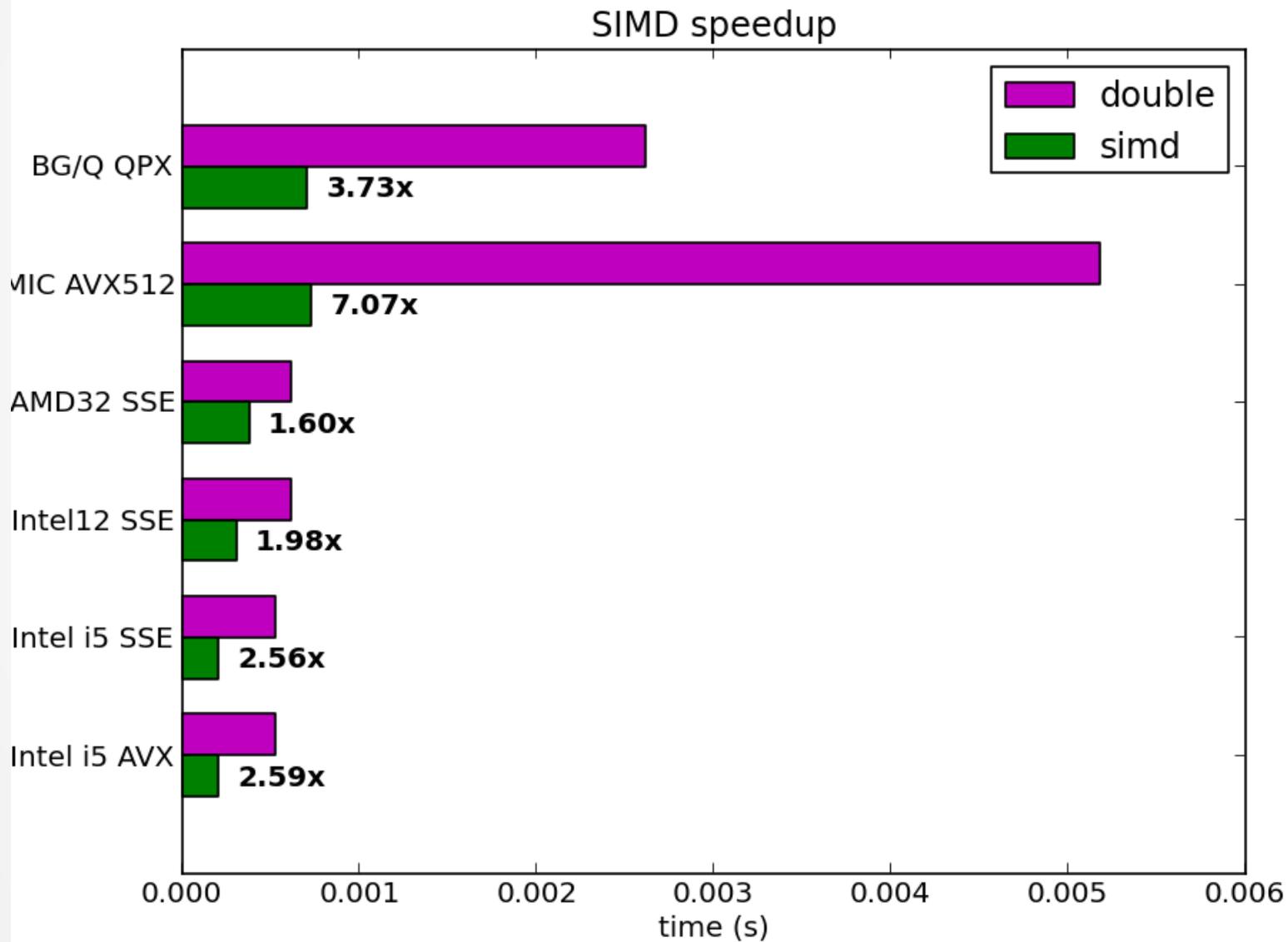
Abstraction Penalty



SIMD implementation

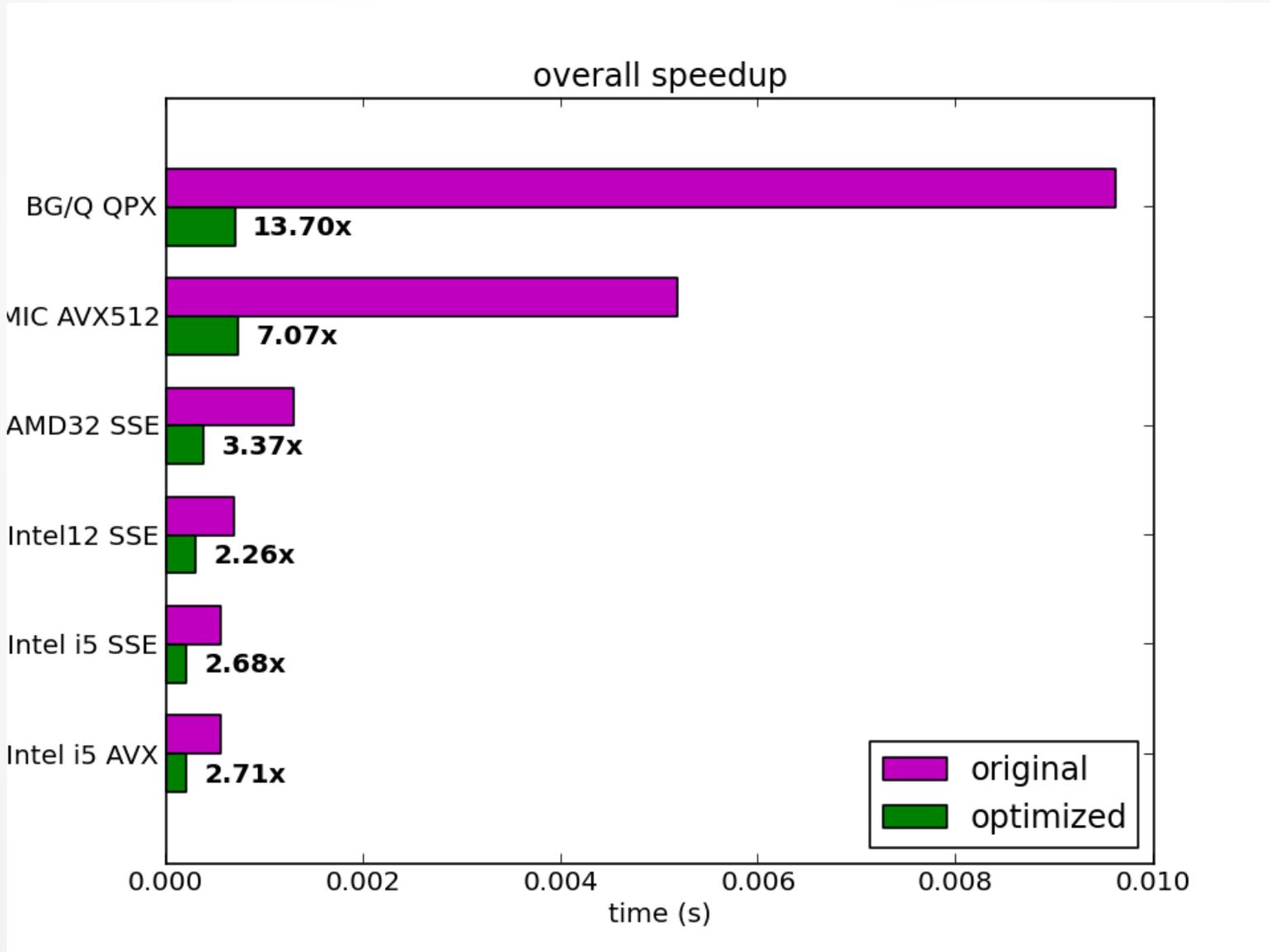
- Use SIMD instructions to work on 2, 4 or 8 particles at once
- Different C++ implementations on different platforms
 - Intel/AMD: vectorclass
 - <http://www.agner.org/optimize/#vectorclass>
 - Great!
 - MIC: mic/micvec.h
 - Provided by Intel
 - Not quite great
 - Blue Gene: vector4double extension
 - Provided by IBM
 - Type and functions, no operator overloading
 - Painful, write-only
- Efficiency requires transpose of particle data
 - Boost MultiArray simply takes a flag

SIMD-enhanced results



Overall Speedup

- Abstraction penalty reduction, SIMD, magic factor



Conclusions

- Accelerator beam dynamics leads to specialized PIC calculations
 - With a wide range of problem sizes
- Current Synergia has excellent scaling characteristics on traditional architectures
- Working toward a production version of Synergia for new generation
 - Leveraging optimization efforts across platforms
 - Optimizing systems large, small and in-between

Backup slides

Hardware

- Intel12: dual-socket, six-core Intel Westmere
- AMD32: quad-socket, eight-core AMD Opteron
- MIC: Intel Xeon Phi 5110P

Drift routine

```
template <typename T>
inline void drift_unit(T& x, T& y, T& cdt, T& xp, T& yp, T& dpop,
                    double length, double reference_momentum, double m,
                    double reference_time) {
    T inv_npz = invsqrt((dpop + 1.0) * (dpop + 1.0) - xp * xp - yp * yp);
    T lxpr = xp * length * inv_npz;
    T lypr = yp * length * inv_npz;
    T D = sqrt(lxpr * lxpr + length * length + lypr * lypr);
    T p = dpop * reference_momentum + reference_momentum;
    T E = sqrt(p * p + m * m);
    T beta = p / E;
    x += lxpr;
    y += lypr;
    cdt += D / beta - reference_time;
}

// T can be double, Vec2d (SSE) , Vec4d (AVX), F64vec8 (~AVX512)
// T can also be (CHEF) Particle::Component or
// JetParticle::Component (polynomial calculation)
```

QPX drift routine

```
inline void drift_unit(vector4double& x, vector4double& y, vector4double& cdt,
    vector4double& xp, vector4double& yp, vector4double& dpop,
    vector4double const& length, vector4double const& reference_momentum,
    vector4double const& m, vector4double const& reference_time) {
    vector4double one = {1,1,1,1};
    vector4double inv_npz = rsqrt4(
        vec_sub(
            vec_sub(
                vec_mul(vec_add(one, dpop),
                    vec_add(one, dpop)),
                vec_mul(xp,xp)), vec_mul(yp,yp)));
    vector4double lxpr = vec_mul(vec_mul(length, xp), inv_npz);
    vector4double lypr = vec_mul(vec_mul(length, yp), inv_npz);
    vector4double D = sqrt4(vec_add(vec_add(vec_mul(length,length),
        vec_mul(lxpr,lxpr)), vec_mul(lypr, lypr)));
    vector4double p = vec_add(reference_momentum,
        vec_mul(reference_momentum, dpop));
    vector4double E = sqrt4(vec_add(vec_mul(p,p), vec_mul(m,m)));
    vector4double beta = vec_sdiv_nochk(p, E);
    x = vec_add(x,lxpr);
    y = vec_add(y,lypr);
    cdt = vec_add(cdt, vec_sub(vec_sdiv_nochk(D, beta),reference_time));
}
```