AMT Runtimes, Portability and the Sunway TiahuLight

Martin Berzins

How hard is it to port an Asynchronous Many Task Runtime code (Uintah) to the worlds fastest machine, the Tiahu SunwayLight ? What does this tell us about the Tiahu SunwayLight?

NSF Grant 1337145 Damodar Sahasbarude Utah.

Zhang Yang IAPM Beijing, Alan Humphrey U. of Utah

Uintah Background and Acknowledgements
DOE NSF People

- DOE ASC Strategic Academic Alliance Program 1998 -2010
- ALCC and Directors Discretionary time awards
- INCITE (4 awards 700M cpu hours in total)
- Argonne and Oak Ridge Facilities
- **NNSA PSAAP2 center funding 2014-2019**
- Argonne A21 exascale early science program
- NSF software funding and Peta-Apps 2007- 2015
- NSF XSEDE TACC computer time and facilities
- The 50 or so people on Uintah and its related projects, since 2003 particularly Justin Luitjens, Qingyu Meng, John Schmidt, Alan Humphrey, John Holmen , Brad Peterson, Steve Parker, Dave Pershing and Phil Smith

Outline

1. Overview of AMTs and Uintah
2. A runtime system portability approach
3. Porting to the Sunway light
4. Summary

Asynchronous Many Task Runtime Systems

SC16 Survey by Thomas Sterling https://www.youtube.com/watch?v=rStVp19tXqk

Darma – Sandia Labs
Legion – Stanford
Charm++ Illinois
Uintah - University of Utah
STAPL – Texas A & M
OCR - Rice
Qthreads – Sandia
LFRIC - UK met Office
PaRSEC - Tennessee
StarPU - Barcelona/INRIA

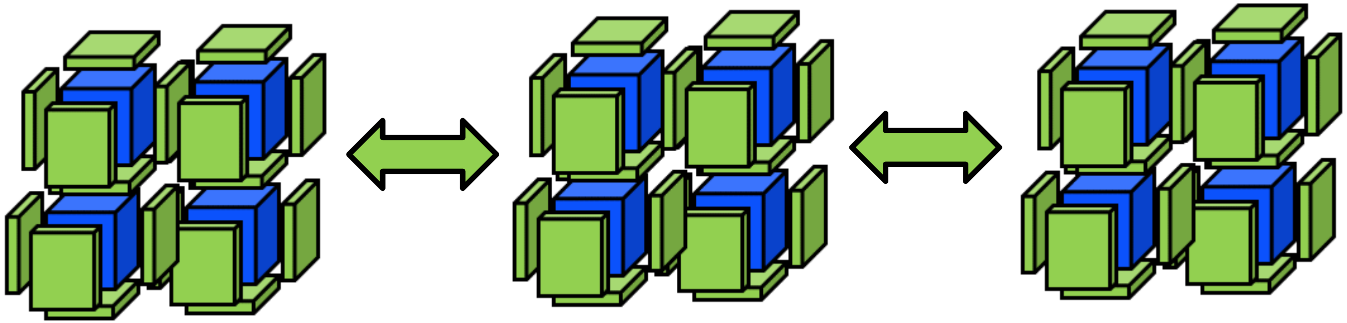
Key features:

Adaptive execution of tasks

Ability to hide communications costs including delays

Task specification may not change as code ported , even though some of runtime does

Spatial Mesh Based Calculations many codes use Bulk Synchronous Parallel (BSP) Processing



Three compute nodes

Synchronize

Compute values on core cells 🟦

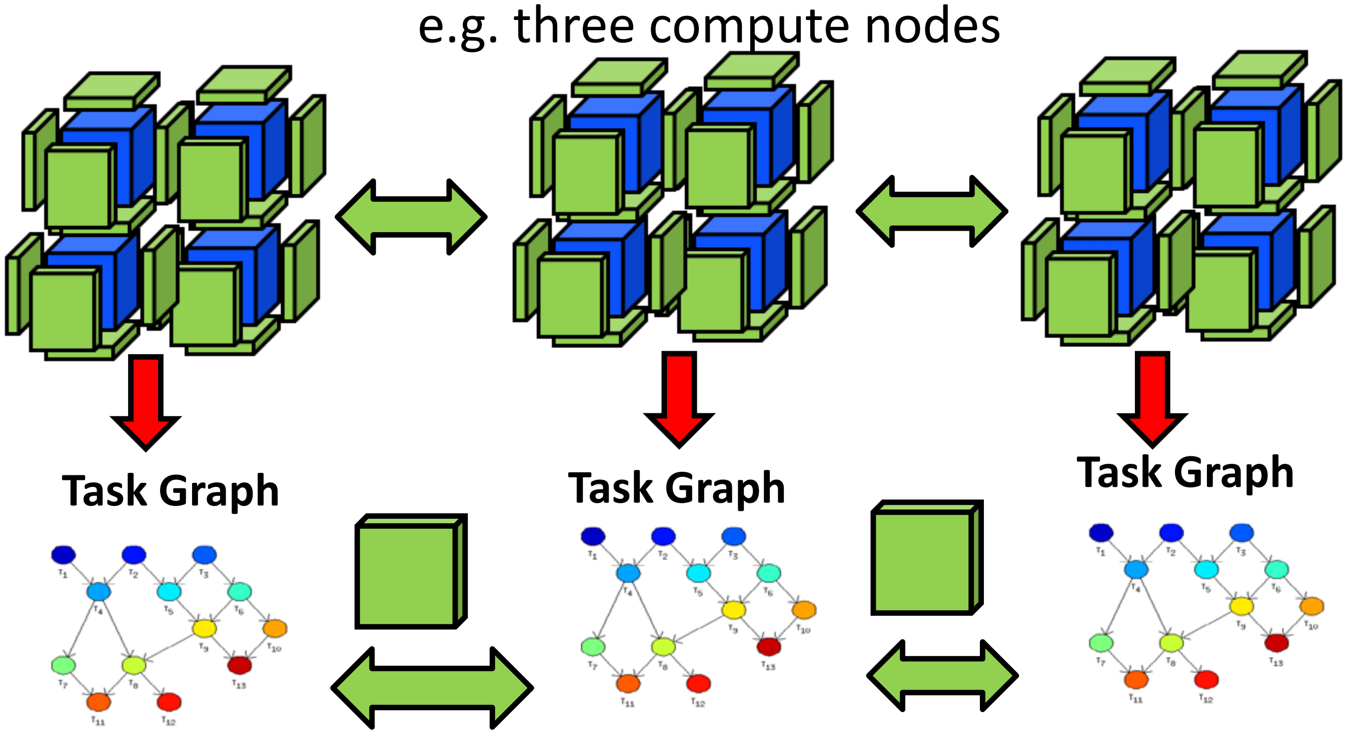
Synchronize

Communicate 🟩 between nodes to update **halo values**

Synchronize

Compute values on core cells 🟦

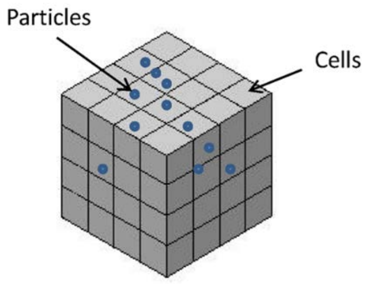
Asynchronous Many Task (AMT) Approach [Sarkar 1987]



Execute tasks when possible communicating as needed. Do useful work instead of waiting

Uintah Architecture

PDE Applications Code Components



Uintah Patch

Components NOT architecture specific written in task form

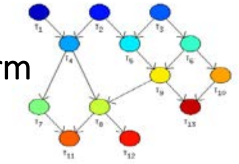
Adaptive execution of tasks

asynchronous out-of-order execution, work stealing, overlap communication & computation.

ICE FV Fluids	50K Lines
MPM Particles	250K lines
ARCHES	250K lines

Task Graph Compilation

Automatically generated abstract C++ task graph form With mpi compiled in

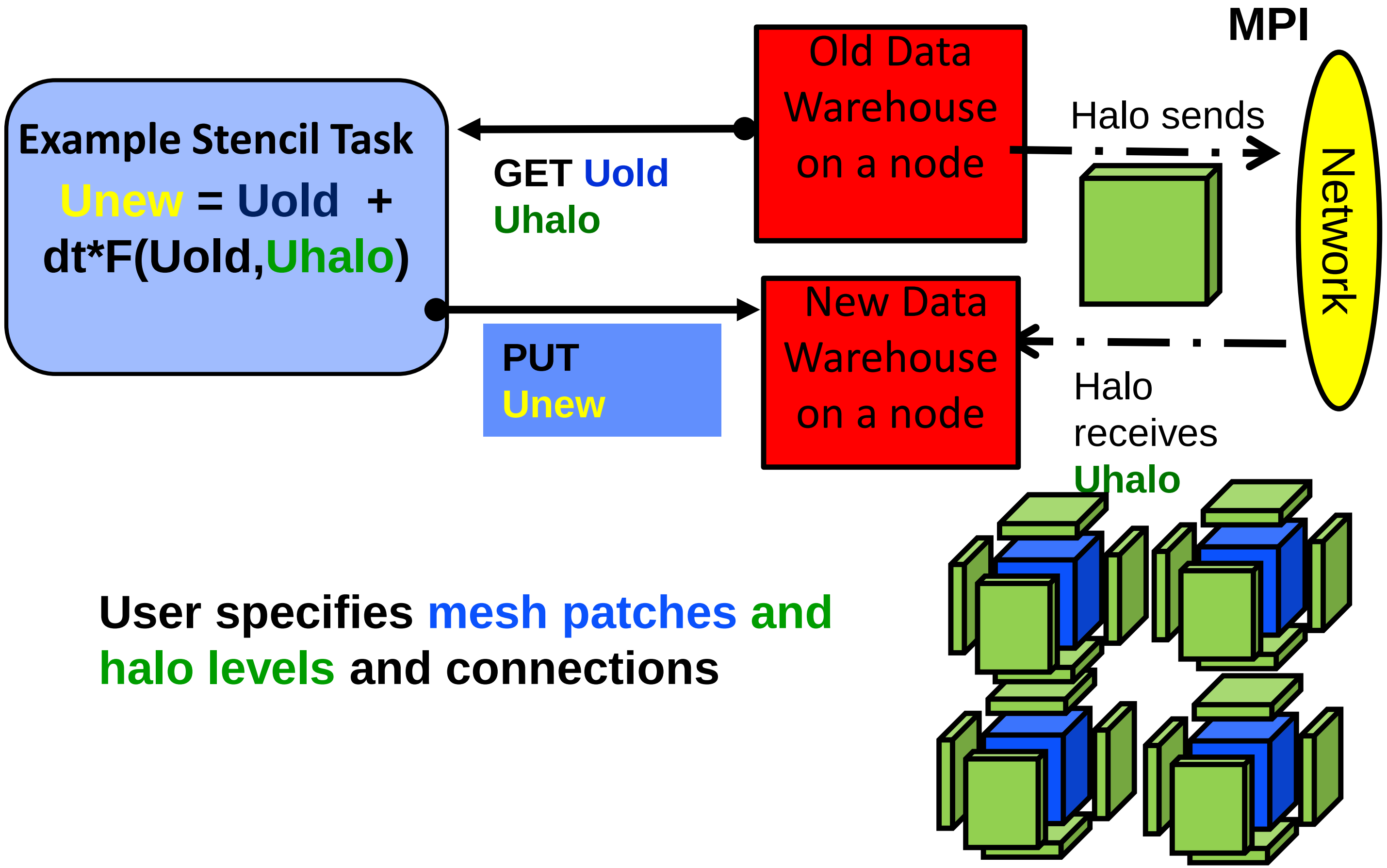


Runtime System

About 1.2 M lines 500K "core' C++
1-2 staff scientists 3-4 students

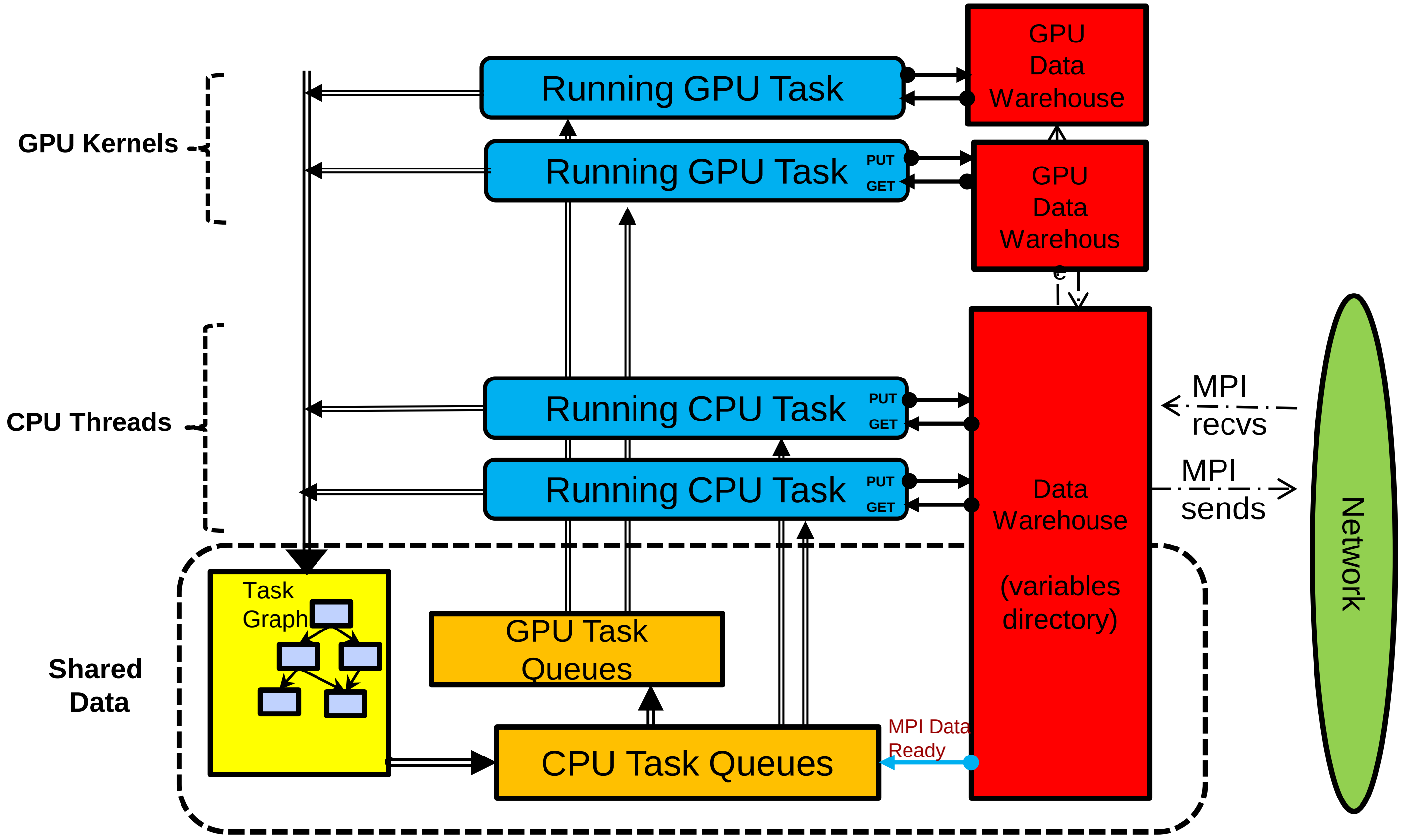
GPUs ***CPUs*** ***Xeon Phis***

Uintah Programing Model for Stencil Timestep [Parker 1998]


MPI
Example Stencil Task
Unew = Uold + dt*F(Uold,Uhalo)
GET Uold Uhalo
PUT Unew
Old Data Warehouse on a node
New Data Warehouse on a node
Halo sends
Halo receives Uhalo
Network

User specifies mesh patches and halo levels and connections

Uintah: Unified Heterogeneous Scheduler & Runtime node [Meng, Luitjens 2012]

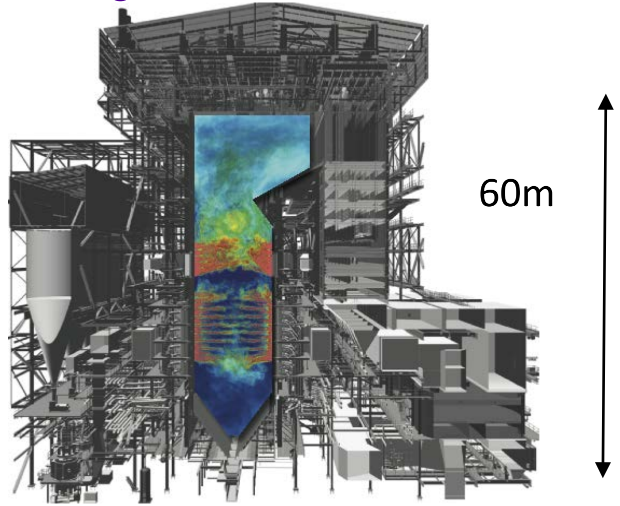


CPU cores and GPUs pull work from task queues

NNSA PSAAP2 Existing Simulations of GE Clean coal Boilers using ARCHES in Uintah

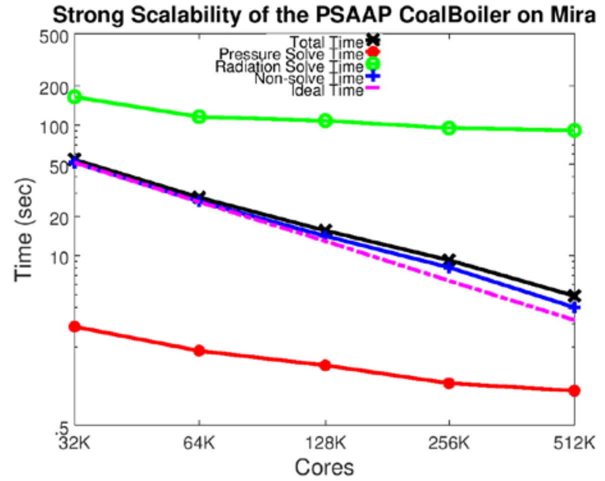
- Large scale turbulent combustion
- Structured, high order finite-volume discretization
- Mass, momentum, energy conservation
- LES closure, tabulated chemistry
- PDF mixing models
- DQMOM (many small linear solves)
- Uncertainty quantification

- **Low Mach number approx. (pressure Poisson solve up to 10^{12} variables**
- **Radiation via Discrete Ordinates – massive solves Mira or ray tracing Titan.**
- **Weak and strong scaling on Mira and strong scaling on Titan.**



Good (superlinear!) scaling for the full problem on Titan and Mira

Cores/GPUs	16k/1k	32k/2k	64k/4k	128k/8k	256k/16k
Time (sec)	821.13	407.31	202.69	99.39	55.06



Porting Uintah to future exascale architectures

Two main tasks

(i) Modify the scheduler to run tasks in a way that takes advantage of the machines in question

(ii) Find a way of transforming the loops in a task so that performance portability is possible

If this works it should work on what might be considered one of the harder machines to port to

Porting Uintah for future exascale architectures

Two main tasks

(i) Modify the scheduler to run tasks in a way that takes advantage of the machines in question

Use experience with many different architectures to do this quickly

(ii) Find a way of transforming the loops in a task so that performance portability is possible

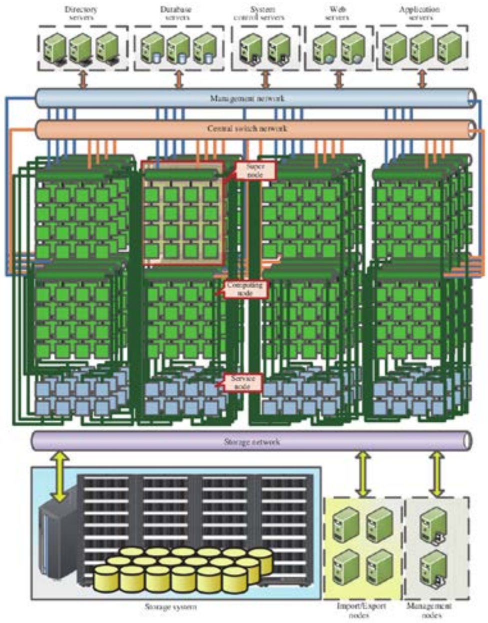
Either hand tune loops or use Kokkos DOE Sandia to have portable loops for which it is possible to get performance for.

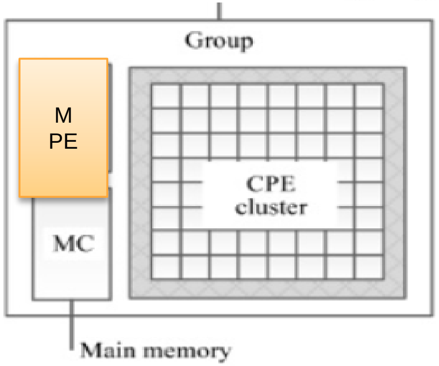
If this works it should work on what might be considered one of the harder machines to port to

Sunway TaihuLight Architecture:

- Each Sunway Compute node contains 4 Compute Processing Elements (CPE).
- 1 CPE consists of 1 Management Processing Element (MPE) and 64 Computing Processing Elements (CPE) along with Memory Controller MC) and System Interface (SI)
- MPE handles the main control flow / management, communications and computations and shares its memory with…..
- CPEs are used to perform computations. These can be considered as “coprocessor” used to offload computations. With 256 vector instructions. Cacheless but with shared scratch memory 64K (LDM)
- 10M cores 93PF vectorization and comms hiding keys to success.

Compute Processing Element (CPE) 4 per node with total of 4 MPEs and 256 cores



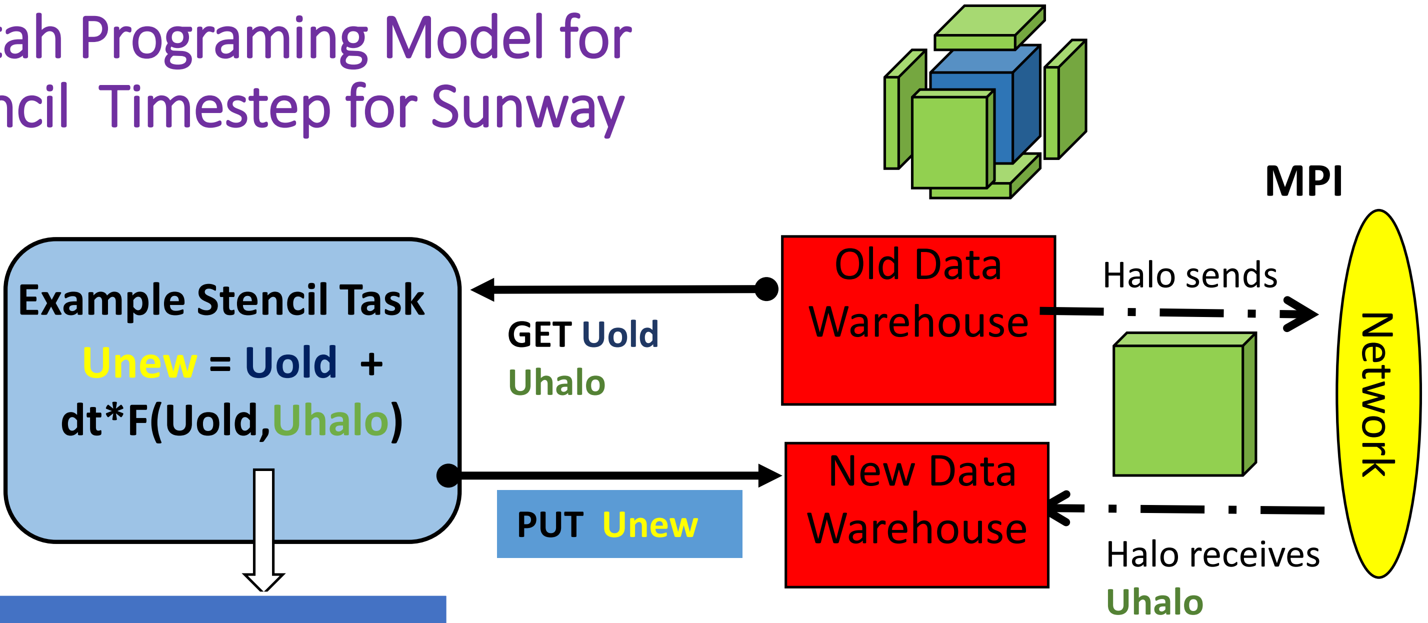


Source https://science.energy.gov/~/media/ascr/ascac/pdf/meetings/201609/Dongarra-ascac-sunway.pdf

Present Applications experience with the Sunway TiahuLight

- Multiple Gordon Bell Finalists in SC16 (3/6) and SC17 (2/3) winner in both years
- Example entry is climate code CAM-SE in SC17 [Haohuan Fu et al.]
  - 152K of 754K lines modified
  - 59K new lines of code
  - Large coding team (12 or more in the team) SC17
- Need to hide data movement to overcome lack of cache and slow communications
- Is there an easier way to do this with a runtime system and fewer programmers?

Uintah Programing Model for Stencil Timestep for Sunway

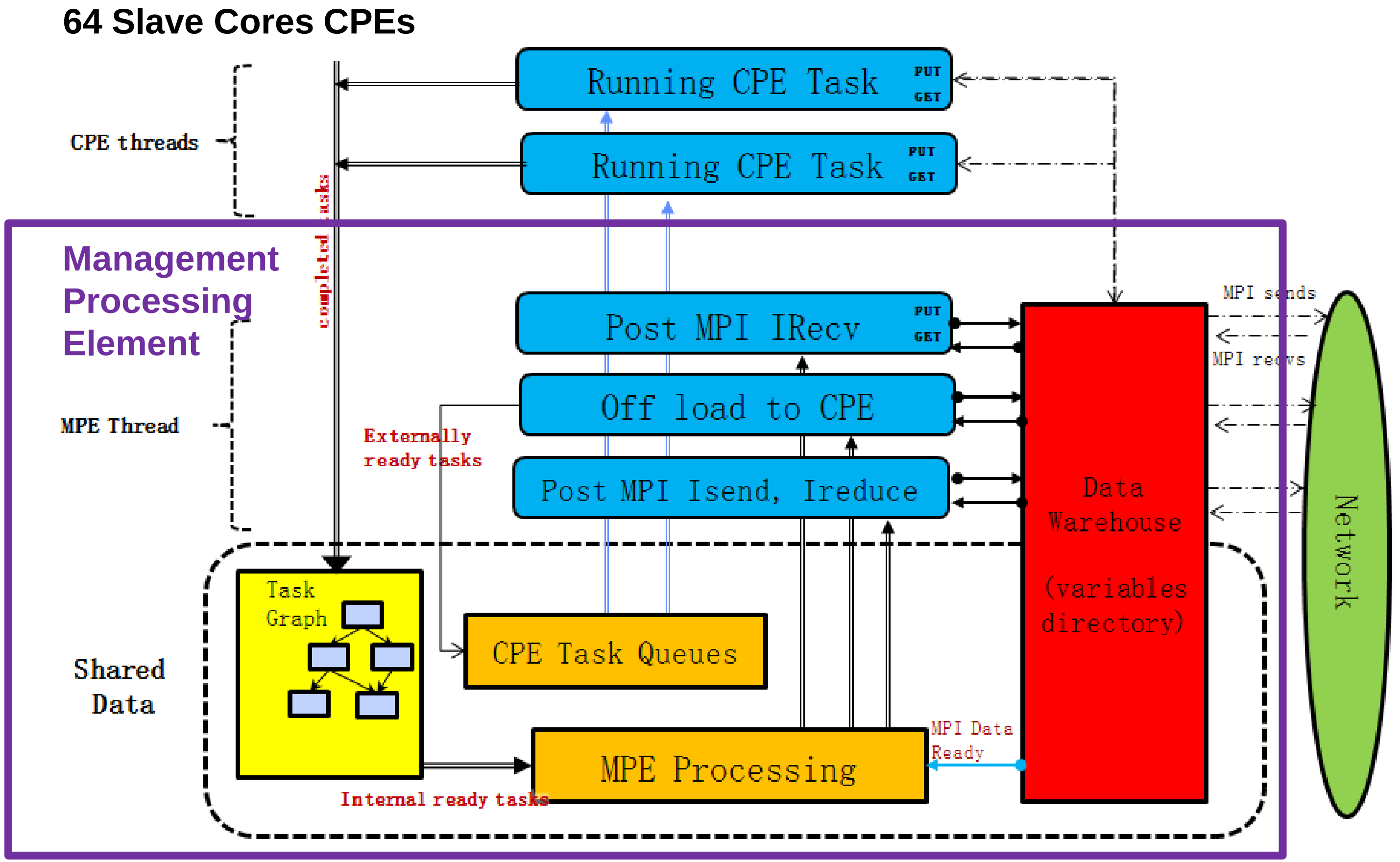


Mixture of C and FORTRAN athreads loops OpenACC also break mesh block into 16x16x8 tiles

$$\frac{\partial u}{\partial t} = a(x,y,z,t)\frac{\partial u}{\partial x} + b(x.y.z,t)\frac{\partial u}{\partial y} + c(x,y,z,t)\frac{\partial u}{\partial z} + \mathrm{c}\left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right]$$

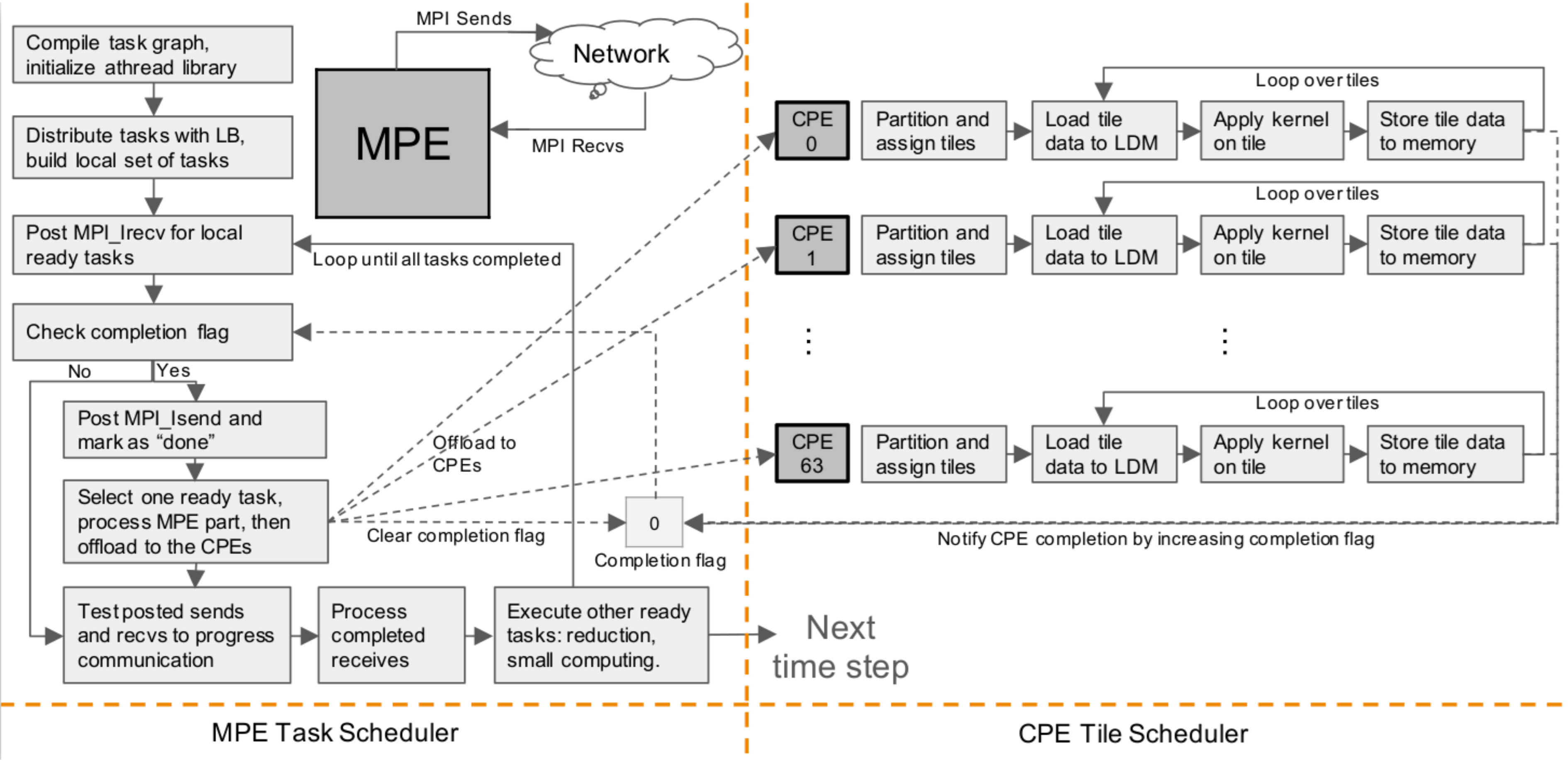
PDE example
modified Burgers equation

Uintah: Asynchronous Heterogeneous Scheduler & Runtime



No MPI inside CPEs lock free DW CPEs pull work

Sunway Runtime System Details


Compile task graph, initialize athread library
Distribute tasks with LB, build local set of tasks
Post MPI_Irecv for local ready tasks
Check completion flag
No
Yes
Post MPI_Isend and mark as “done”
Select one ready task, process MPE part, then offload to the CPEs
Test posted sends and recvs to progress communication
Process completed receives
Execute other ready tasks: reduction, small computing.
MPE
MPI Sends
Network
MPI Recvs
Loop until all tasks completed
Offload to CPEs
Clear completion flag
0
Completion flag
CPE 0
CPE 1
CPE 63
Partition and assign tiles
Load tile data to LDM
Apply kernel on tile
Store tile data to memory
Loop over tiles
Notify CPE completion by increasing completion flag
Next time step
MPE Task Scheduler
CPE Tile Scheduler

200 new lines of code

200 new lines of code

Sunway specific changes

Infrastructure and Scheduler:

- GNU compiler used for MPE infrastructure code written in C++11
- Updated offloading and polling mechanism using OpenACC

Computational Kernel / Task:

- **Porting of Kernel**: -main comp. kernel rewritten using Fortran, C, OpenACC and native athread runtime as CPEs do not support C++
- Need to use athread to overcome OpenACC slowdowns [SC17 Gordon Bell CAM paper]
- **Optimizations**:
  - ***Tiling***: Used tiling for efficient use of LDM for stencil like operations in Burgers' equation. The CPE part of scheduler divides tiles among CPEs.
  - ***Vectorization***: Used native SIMD vector intrinsics for vectorization as Sunway toolchains do not support automatic vectorization also used fast math library for faster exp calculations, as no h/w suppport

Original C++ kernel for Burgers' equation

Fortran kernel vectorized with SIMD intrinsics for d2udz2 calculation

```fortran
VECTOR256 :: v0, v1, v2, v3, v_d2udz2
! d2udz2 = (-2 * u0(i,j,k) +u0(i,j,k-1) &
!           + u0(i,j,k+1)) * (z_dx*z_dx)
v0 = SIMD_CMPLX(-2d0, -2d0, -2d0, -2d0)
call SIMD_LOADU(v1, u0(i,j,k))
call SIMD_LOADU(v2, u0(i,j,k-1))
call SIMD_LOADU(v3, u0(i,j,k+1))
v0 = SIMD_VMAD(v0, v1, v2)
v0 = SIMD_VADDD(v0, v3)
tmp2 = z_dx * z_dx
call SIMD_LOADE(v2, tmp2)
v_d2udz2 = SIMD_VMULD(v0, v2)
```

```cpp
pragma omp parallel for collapse(2)
   for(k=a; k<b; ++k)
   {
       for(j=c; j<d; ++j)
       {
pragma simd
           for(i=e; i<f; ++i)
           {
               double px = anc0 + c0 * i, py = anc1 + c1 * j, pz = anc2 + c2 * k;

               double dudx = -1 * (u_data[k][j][i] - u_data[k][j][i - 1]) * initialW(px, currTime) /dx_x;
               double dudy = -1 * (u_data[k][j][i] - u_data[k][j - 1][i]) * initialW(py, currTime) /dx_y;
               double dudz = -1 * (u_data[k][j][i] - u_data[k - 1][j][i]) * initialW(pz, currTime) /dx_z;

               double d2udx2 = (-2 * u_data[k][j][i] + u_data[k][j][i - 1] + u_data[k][j][i + 1]) /(dx_x * dx_x);
               double d2udy2 = (-2 * u_data[k][j][i] + u_data[k][j - 1][i] + u_data[k][j + 1][i]) /(dx_y * dx_y);
               double d2udz2 = (-2 * u_data[k][j][i] + u_data[k - 1][j][i] + u_data[k + 1][j][i]) /(dx_z * dx_z);

               double du = dt * ((dudx + dudy + dudz) + (double)VISCOSITY*(d2udx2 + d2udy2 + d2udz2));

               new_u_data[k][j][i] = u_data[k][j][i] + du;
           }//for(i=ti; i<ti_max; ++i)
       }//for(j=tj; j<tj_max; ++j)
   }//for(k=tk; k<tk_max; ++k)
```

Sunway - Strong scaling Results

From 1 to 128 nodes. Modes:

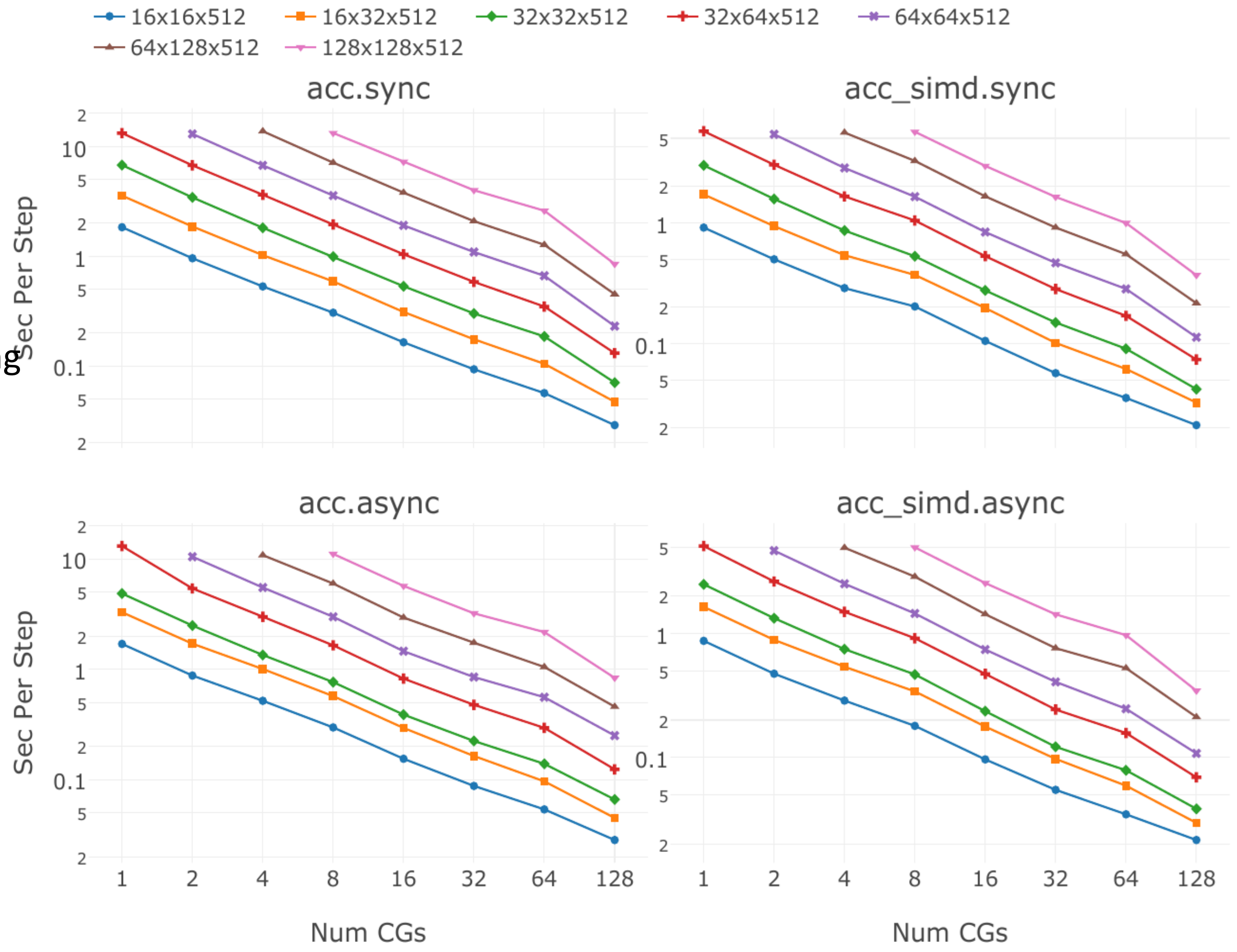
Acc.sync: sync. scheduling

acc_simd.sync: sync. scheduling using simd instructions

acc.async: async. scheduling

acc_asimd.sync: async. scheduling using simd instructions

Maximum improvement of async version over sync is about 22%



Results

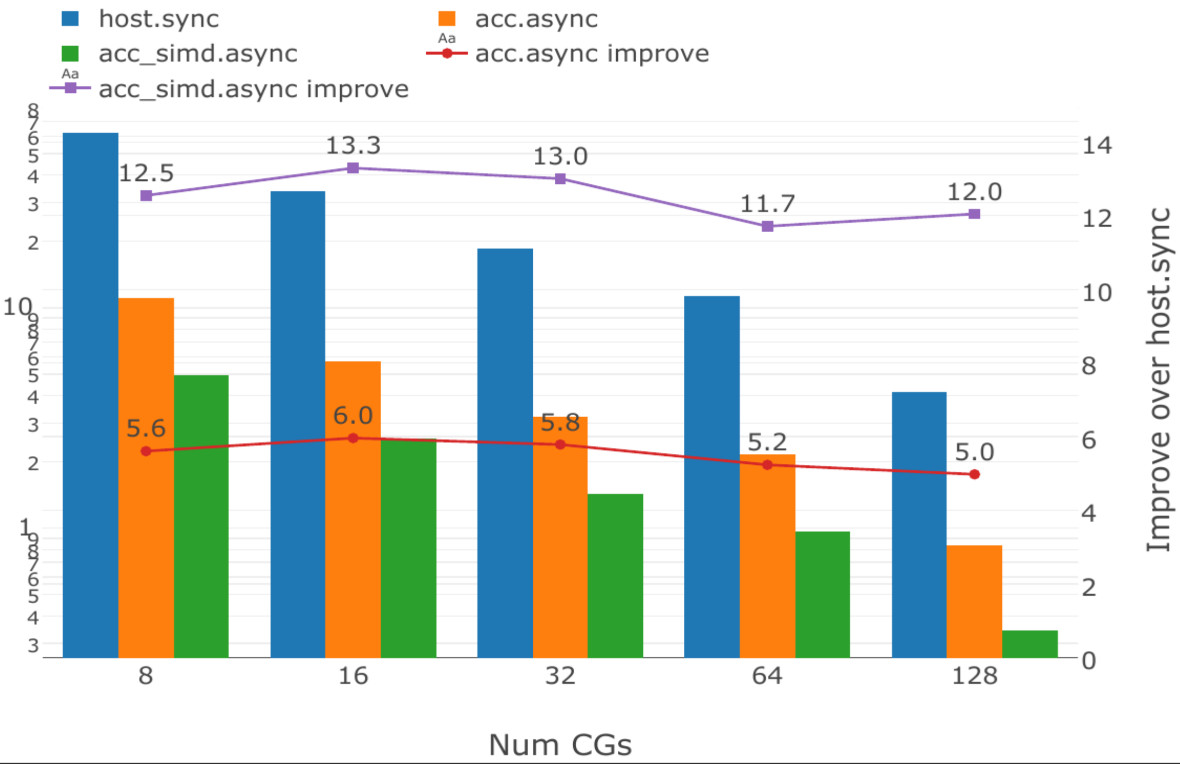
Strong scaling with up to 90% efficiency up to 8K cores (limit of test queue)

Peak performance only 1% of peak but 1 TF

Asynchronous scheduler give 6X speedup

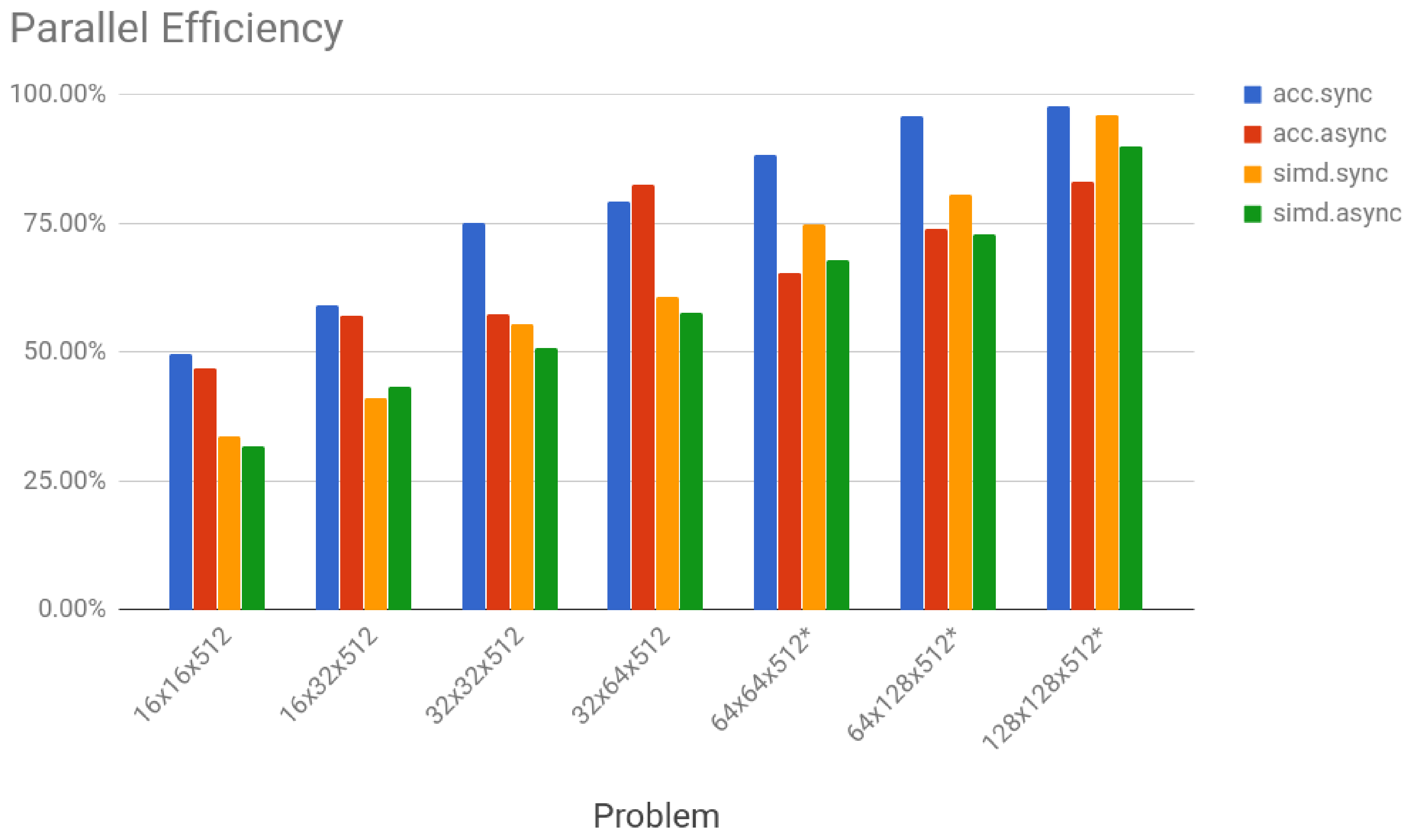
SIMD instructions give 13X speedup

Baseline is host.sync - most naive implementation on Sunway: synchronous execution without offloading to CPEs.



Sunway - Parallel Efficiency

> 90% for sufficiently large problem



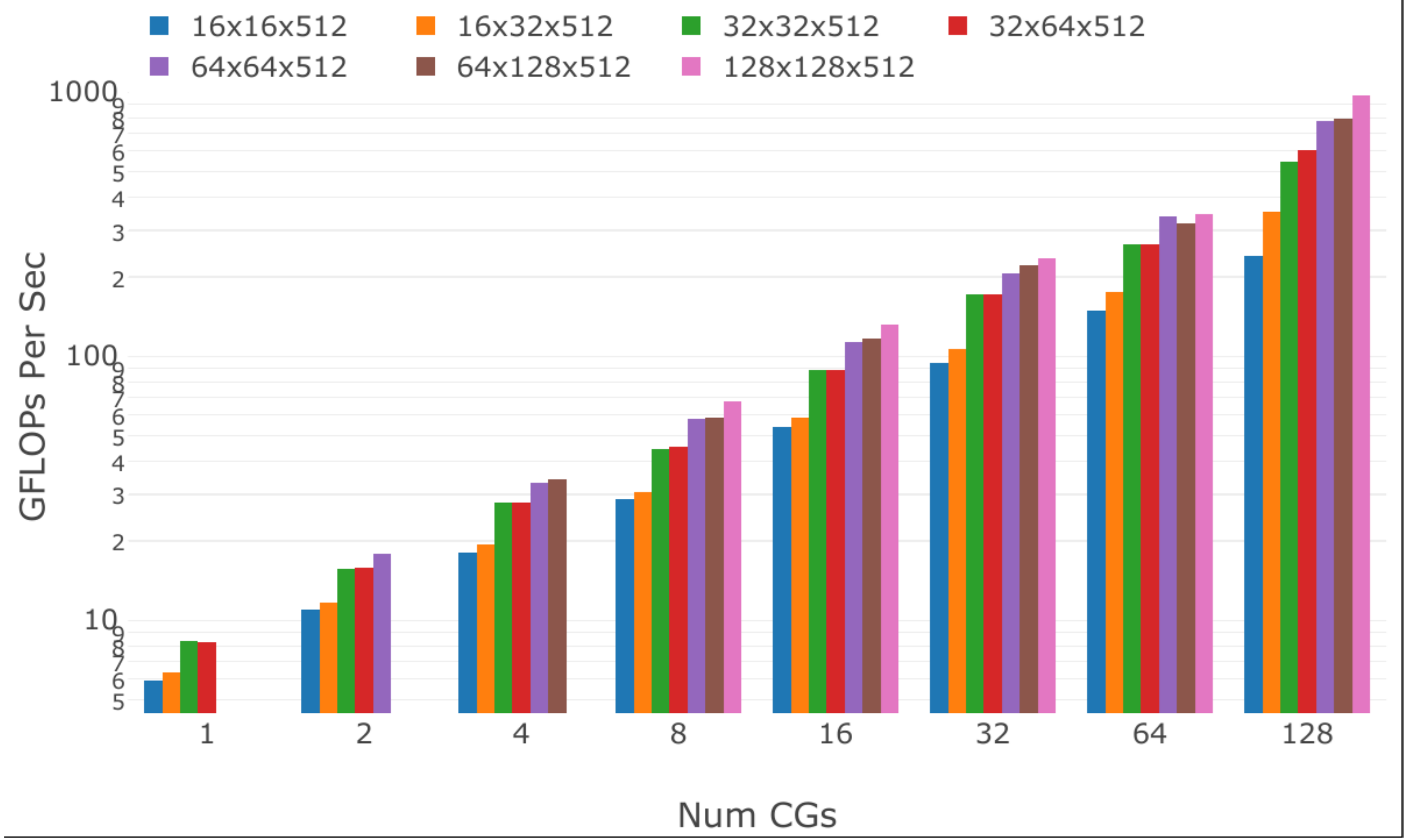
Summary

- Porting strategy works well – good scaling and reasonable peak
- 2 person months of work to show how 1M lines of code might run
- **New scheduler for Sunway required only 400 lines of code**
- Sunway software environment is still at an early stage, e.g. first GPUs
- Tasks needed to be hand rewritten in C, athreads and Fortran
- Porting a full Uintah package with 500-600 loops would be a daunting task
- Use of a portability library like Kokkos would solve this problem – does not exist
- Our experiences with OpenACC similar to CAM climate code Gordon Bell SC17 paper

Additional Slides

Sunway - GFLOPS

Because of memory bound nature of Burgers code, it achieved only 1% of peak GFLOPS.



Uintah Grid Spec - Load Balancer

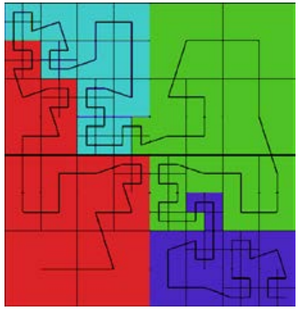
```
<Grid>
    <Level>
       <Box label = "1">
       <lower>       [0,0,0]       </lower>
       <upper>       [1.0,1.0,1.0] </upper>
       <resolution> [64,64,64]    </resolution>
       <patches>     [2,2,2]       </patches>
       <extraCells> [1,1,1]       </extraCells>
     </Box>
   </Level>
 </Grid>
```

XML Problem Specification provided by user input file, e.g.

- e.g., Grid spec, patches
- <patches> [x,y,z] </patches>

Uintah Load Balancer automatically assigns patches to nodes

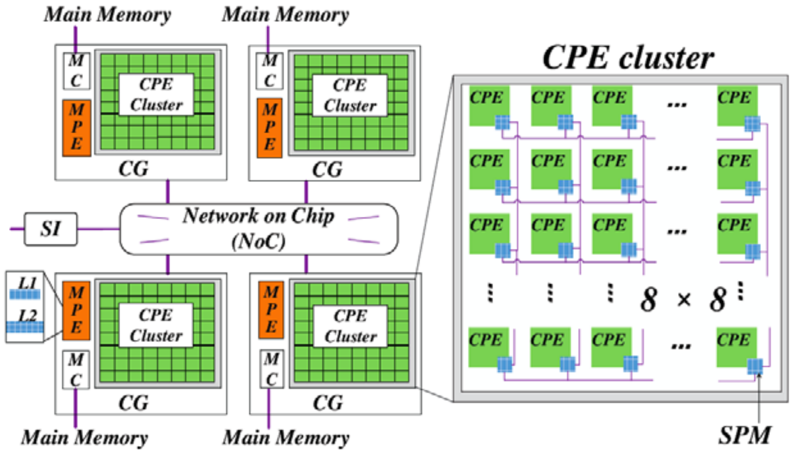
- **Improved Load Balancing**
  - Cost Estimation Algorithms based on data assimilation
  - Use load balancing algorithms based on patches and a fast space filling curve algorithm [1]

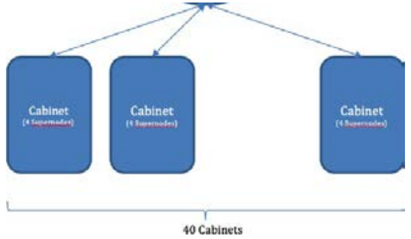


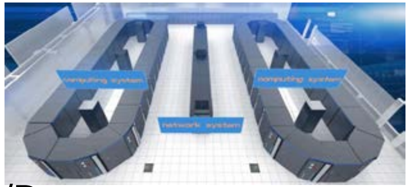
Source Jack Dongarra

Sunway TaihuLight

- SW26010 processor,Chinese design, fab, 1.45 GHz
- Node = 260 Cores (1 socket)
  - 4 – core groups
    - 64 CPE, No cache, 64 KB scratchpad/CG
    - 1 MPE w/32 KB L1 dcache & 256KB L2 cache
  - 32 GB memory total, 136.5 GB/s
  - ~3 Tflop/s, (22 flops/byte)
- Cabinet = 1024 nodes
  - 4 supernodes=32 boards(4 cards/b(2 no
  - ~3.14 Pflop/s
- 40 Cabinets in system
  - 40,960 nodes total
  - 125 Pflop/s total peak
- 10,649,600 cores total
- 1.31 PB of primary memory (DDR3)
- 93 Pflop/s for HPL, 74% peak
- 0.32 Pflop/s for HPCG, 0.3% peak
- 15.3 MW, water cooled
  - 6.07 Gflop/s per Watt
- 1.8B RMBs ~ $280M, (building, hw, apps, sw, …)







https://science.energy.gov/~/media/ascr/ascac/pdf/meetings/201609/Dongarra-ascac-sunway.pdf

SW26010 Processor

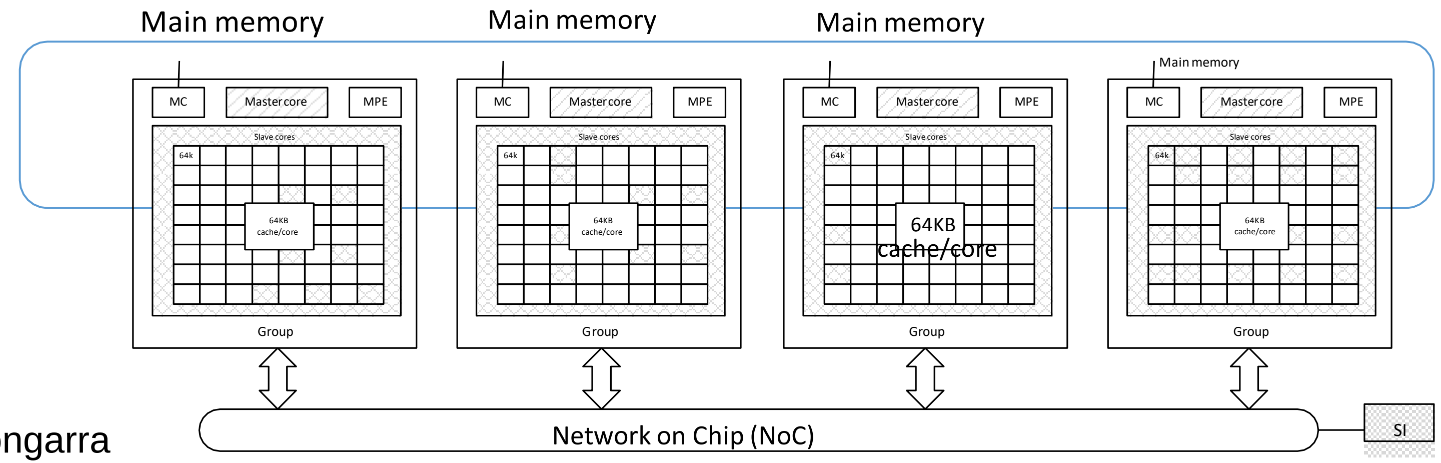
- China's first homegrown many-core processor
  - Vendor: Shanghai High Performance IC Design Center
  - supported by National Science and Technology Major Project (NMP): Core Electronic Devices, High-end Generic Chips, and Basic Software
  - 28 nm technology
  - 260 Cores
  - 3 Tflop/s peak

Source Jack Dongarra

https://science.energy.gov/~/media/ascr/ascac/pdf/meetings/201609/Dongarra-ascac-sunway.pdf

SW26010: General Architecture

- 1 node   with 260 cores per processor has 4 Core Groups (CGs), each of which has
  - 1 Management Processing Element (MPE) handles the main control flow / management, communications and computations and shares its memory with…..
  - 64 (8x8) Computing Processing Element(CPE) These can be considered as "coprocessor" used to offload computations. With 256 vector instructions. Cacheless but with shared scratch memory 64K (LDM)



Source Dongarra