Stony Brook University

TOTAL

Evaluating Performance of OpenMP Tasks in a Seismic Stencil Application

Eric Raut[1], Jie Meng[2], Mauricio Araya-Polo[2], Barbara Chapman[1]

[1] Stony Brook University, Stony Brook, NY, USA

[2] Total EP R&T, Houston, TX, USA

IWOMP 2020

September 22, 2020

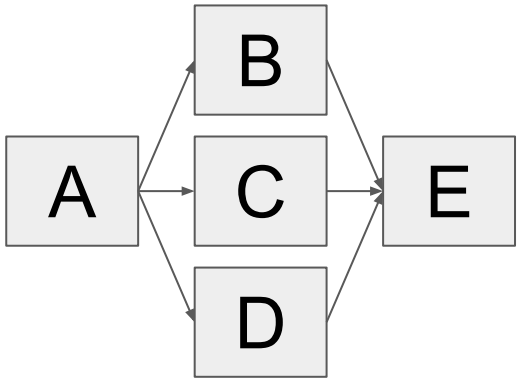
iACS INSTITUTE FOR ADVANCED COMPUTATIONAL SCIENCE

Introduction and Motivation

- Goals
  - Port an existing application from loop-based to task-based approach
  - Investigate performance portability of OpenMP tasks on different architectures
- Our application: *Minimod*
  - Stencil-based application which solves wave equation
- Experiments
  - Performance comparison of loop- and task-parallelism
  - Experiments across architectures: Intel and POWER

Task-based Programming

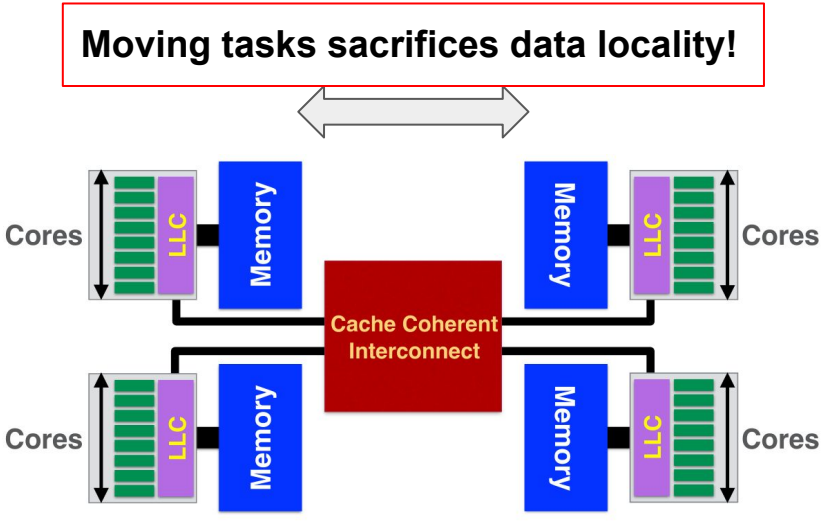
- Program represented as units of work called tasks
- Alternative to loop parallelism
- Multiple tasks can be run in parallel
- Computation represented as a directed acyclic graph (DAG)
- Runtime responsible for running eligible tasks in parallel
- Distributed task-based programming models exist as well
  - PaRSEC, Charm++, Legion, HPX, StarPU, etc.



Example DAG

Challenges

- Load balancing vs. locality tradeoff
- Task granularity tradeoff
- Writing task-based code
- Schedulers
- Performance portability
- Ease of programming and porting



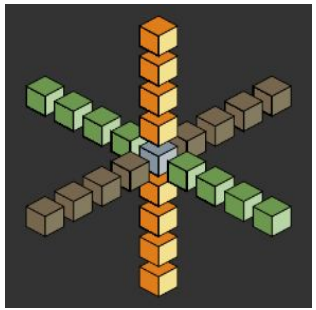
From van der Pas

Few large-scale programs are written using task-based programming!

Application: Geophysics Exploration

- Wave equation important to many geophysics applications
- Computationally intensive part is stencil computation

$$\frac{1}{v_p^2}\frac{\partial^2 p(\mathbf{x},t)}{\partial t^2} - \nabla^2 p(\mathbf{x},t) = f(\mathbf{x},t),$$



- *Minimod*: wave propagation mini-app developed by Total
  - Designed to test new and emerging programming models and hardware

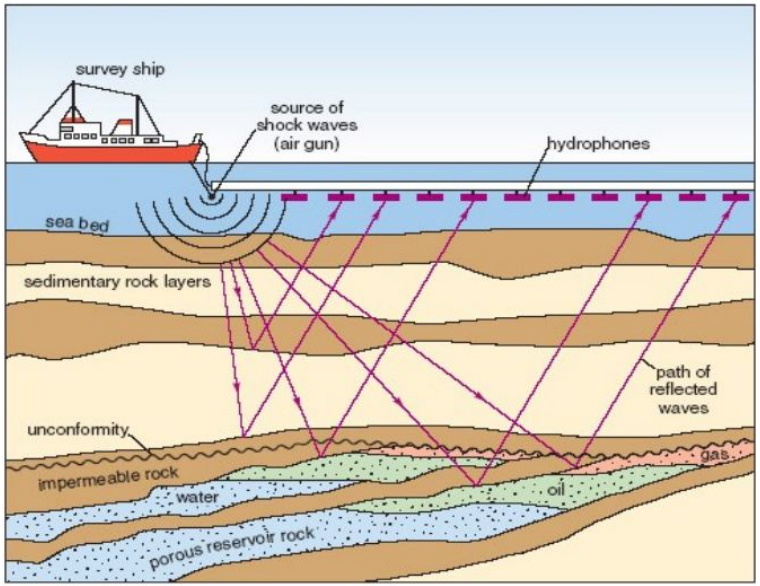


Image from Beaude

Minimod: High-level Schematic

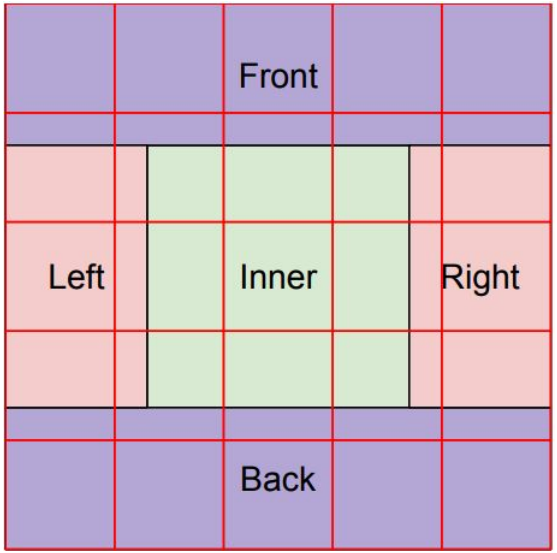
Parallelizable

```
   Data: f: source
   Result: u^n: wavefield at timestep n, for n ← 1 to T
1  u^0 := 0;
2  for n ← 1 to T do
3  |   for each point in wavefield u^n do
4  |   |   Solve Eq. 2 (left hand side) for wavefield u^n;
5  |   end
6  |   u^n = u^n + f^n (Eq. 2 right hand side);
7  end
```

Minimod: Wavefield Solution

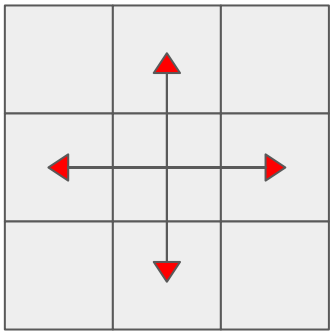
```
   Data: u^{n-1}, u^{n-2}: wavefields at previous two timsteps
   Result: u^n: wavefield at current timestep
 1 for i ← xmin to xmax do
 2     if i ≥ x3 and i ≤ x4 then
 3         for j ← ymin to ymax do
 4             if j ≥ y3 and j ≤ y4 then
 5                 // Bottom Damping (i, j, z1...z2)
 6                 // Inner Computation (i, j, z3...z4)
 7                 // Top Damping (i, j, z5...z6)
 8             else
 9                 // Back and Front Damping (i, j, zmin...zmax)
10             end
11         end
12     else
13         // Left and Right Damping (i, ymin...ymax, zmin...zmax)
14     end
15 end
```

Inherent load imbalance due to boundary conditions



Blocks contain both inner and boundary calculations

Code for OpenMP Tasks and Dependencies



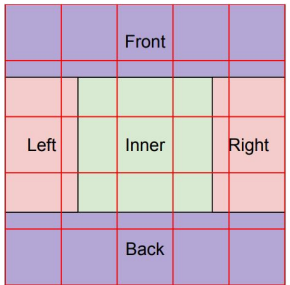
```
#pragma omp task \
    depend(in:    u[IDX3_l(txmin,     tymin,      -lz)]) \ Self, t-1
    depend(in:    u[IDX3_l(txmin_prev,tymin,      -lz)]) \ Left, t-1
    depend(in:    u[IDX3_l(txmin_next,tymin,      -lz)]) \ Right, t-1
    depend(in:    u[IDX3_l(txmin,     tymin_prev, -lz)]) \ Below, t-1
    depend(in:    u[IDX3_l(txmin,     tymin_next, -lz)]) \ Above, t-1
    depend(inout: v[IDX3_l(txmin,     tymin,      -lz)])   Self, t
{
    ...
}
```

We would like dependencies to be more fine-grained than this

Configurations of Minimod Evaluated

- Loop x static/dynamic/guided
- Loop xy (blocking in x and y dimensions) static/dynamic/guided
  - Blocking in x and y dimensions
  - collapse(2) clause used on loops over x and y blocks
- Tasks xy
- Tasks xy “nodep”
  - No dependencies specified
  - Bulk task synchronization point at the end of each timestep

```
   Data: u^{n-1}, u^{n-2}: wavefields at previous two timsteps
   Result: u^n: wavefield at current timestep
 1 for i ← xmin to xmax do
 2 |   if i ≥ x3 and i ≤ x4 then
 3 |   |   for j ← ymin to ymax do
 4 |   |   |   if j ≥ y3 and j ≤ y4 then
 5 |   |   |   |   // Bottom Damping (i, j, z1...z2)
 6 |   |   |   |   // Inner Computation (i, j, z3...z4)
 7 |   |   |   |   // Top Damping (i, j, z5...z6)
 8 |   |   |   else
 9 |   |   |   |   // Back and Front Damping (i, j, zmin...zmax)
10 |   |   |   end
11 |   |   end
12 |   else
13 |   |   // Left and Right Damping (i, ymin...ymax, zmin...zmax)
14 |   end
15 end
```



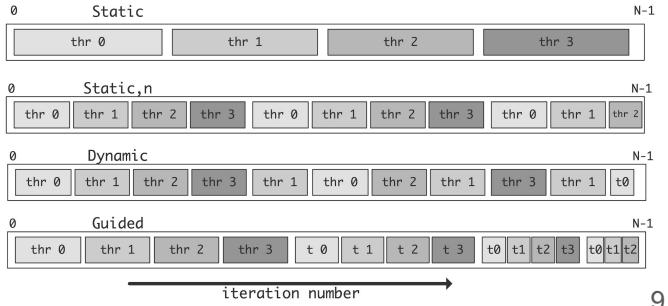
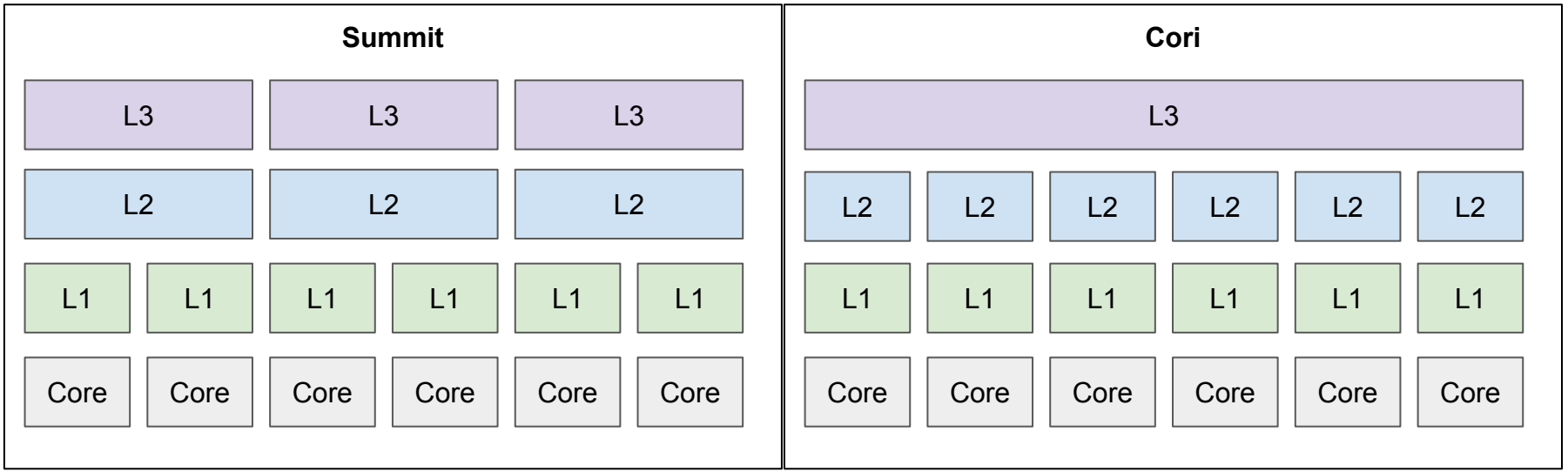


Image from Eijkhout

Experimental Setup

- Summit: Two IBM POWER9 processors: 42 compute cores per node
- Cori: Two Intel Xeon processors: 32 cores per node
- SeaWulf: Two Intel Xeon Gold processors: 40 cores per node



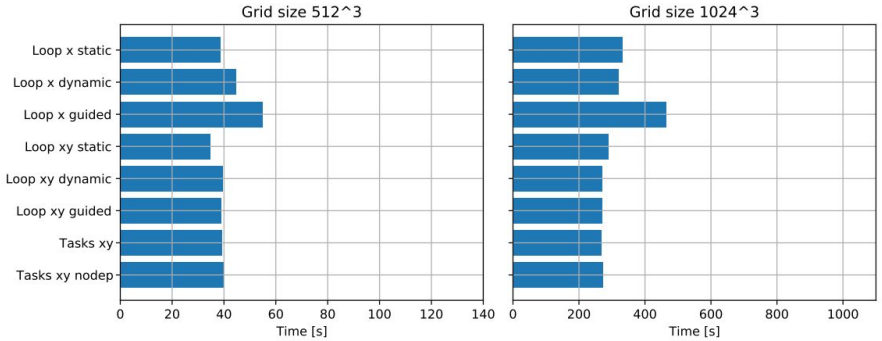
Compilers

- Summit
  - LLVM 9.0
- Cori
  - LLVM 10.0
- SeaWulf
  - LLVM 11.0 (git 3cd13c4)
- Compiler flags
  - -O3 -march=native -fopenmp

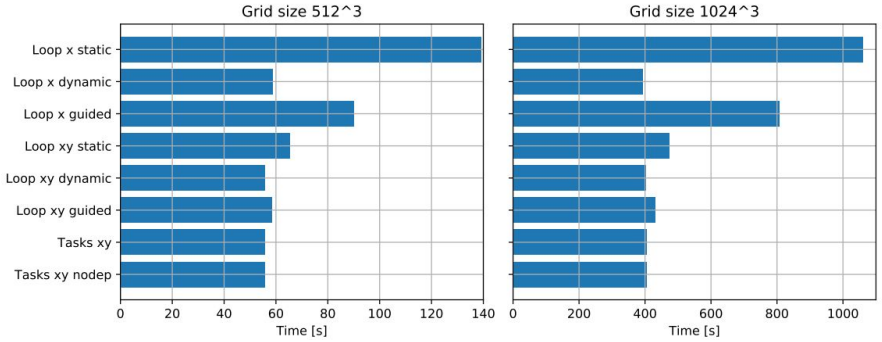
Note:
Summit access provided by OLCF (Oak Ridge National Laboratory)
Cori access provided by NERSC (Department of Energy)
SeaWulf access provided by IACS (Stony Brook University)

Result: Execution Time for each Configuration

Summit



Cori



- Large benefit from xy blocking
- Tasks are generally competitive with loops.
- Summit: benefit from static scheduling at grid 512^3, not 1024^3.
- Cori: large benefit from dynamic and guided, especially for loop x
- In all cases, tasks perform similarly to dynamic scheduled loops

Result: Cache Usage

	Grid size 512^3			**Grid size** 1024^3		
	Summit	Cori	SeaWulf	Summit	Cori	Seawulf
Loop x static	16	19	49	12	16	50
Loop x dynamic	42	10	27	35	9	28
Loop x guided	45	15	41	36	15	47
Loop xy static	9	11	47	7	12	43
Loop xy dynamic	26	11	45	27	11	43
Loop xy guided	26	12	47	22	12	44
Tasks xy	27	12	45	27	11	43
Tasks xy nodep	27	11	45	26	11	43
Average	27.3	12.6	43.3	24.0	12.1	42.6

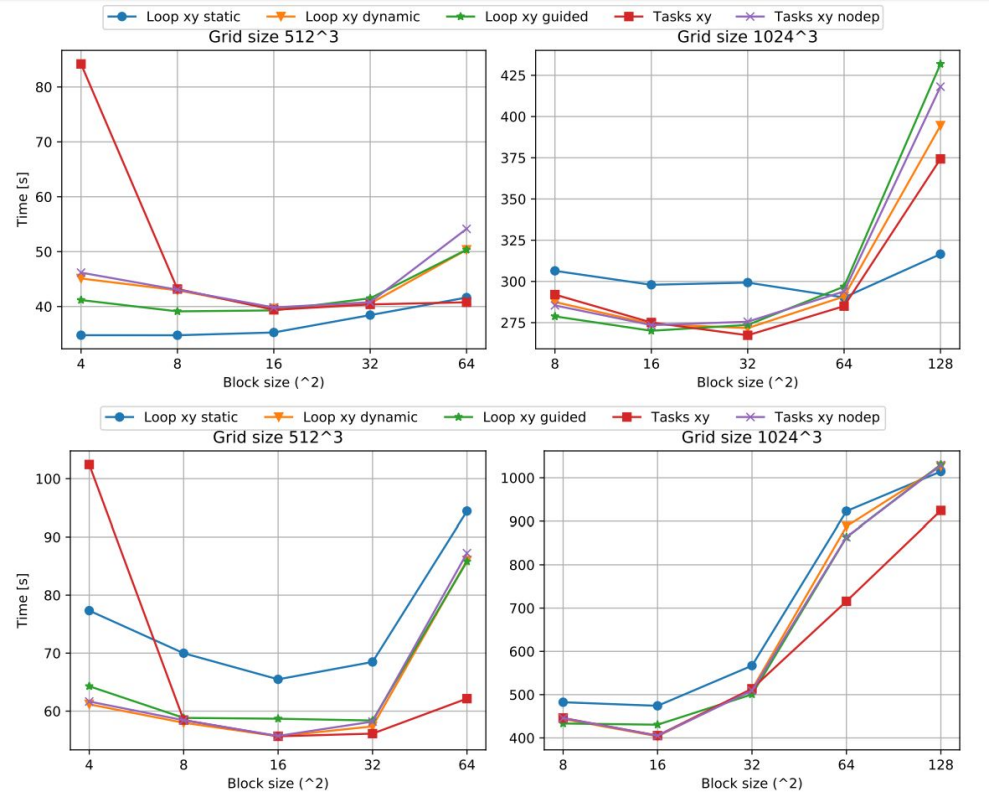
Table 2. L3 miss rate [%] on each computer for each configuration.

- Increased L3 misses from dynamic/guided and tasks on Summit (POWER) compared to static
  - Same L3 miss penalty not seen on Intel
  - Likely due to L3 being split on POWER
- Decreased L3 misses from dynamic (for x loops)

Result: Block Size Sensitivity

Summit

Cori



- Tasks with dependencies see a large spike in execution time at extremely small block sizes
- Tasks without dependencies do not incur this penalty, indicating that the overhead associated with task dependencies may be significant
- Fine-grained task dependencies result in improved performance over tasks without dependencies at larger block sizes

Additional Observations

- Limitations on OpenMP depend clause
  - Overlapping depends regions
- Architecture-dependent scheduling parameters
  - Effect of cache structure on locality
- Lack of NUMA awareness
  - `affinity` clause from OpenMP 5.0 could help alleviate this

Related Work

- Scheduling of tasks well studied from theoretical perspective
  - Practical considerations are less understood
- *Evaluation of OpenMP Task Scheduling Strategies* (Duran et al., IWOMP 2008)
  - Evaluated different scheduling strategies in early implementation of OpenMP
- *SLATE* explores using OpenMP tasks for linear algebra routines
  - Gates et al., SC 2019
- Other (mini) applications have been ported to OpenMP tasks
  - Irregular fast multipole method application (Atkinson et al., IWOMP 2017)
  - AMR proxy application (Rico et al., IWOMP 2019)

Conclusions and Future Work

- Tasks are competitive with loop parallelism, even for this relatively-regular stencil application (and even better in some cases)
- Movement of tasks between cores has a high impact on LLC miss rate (for POWER architecture)
  - Stresses importance of locality-aware task scheduling
  - Suggests optimal scheduling policies may be architecture-dependent
- Task scheduling overhead is high at very small task sizes
  - May be due to dependency analysis
- Future work
  - Adding more kernels
  - OpenMP extensions for tasks on GPUs
  - Evaluating cluster-level task-based programming models

Backup

Minimod

- Fortran (originally) and C benchmark application that solves the 3D acoustic wave equation
- PML damping at the boundary

Finite difference 3D stencil

$$\frac{1}{v_p^2}\frac{\partial^2 p(\mathbf{x},t)}{\partial t^2} - \nabla^2 p(\mathbf{x},t) = f(\mathbf{x},t),$$

$$p^{n+1} - Qp^n + p^{n-1} = \left(\Delta t^2\right) v_p^2 f^n,$$

$$Q = 2 + \Delta t^2 v_p^2 \nabla^2.$$

$$\nabla^2 p(x,y,z) \approx \sum_{m=1}^{4} c_{xm}\left[p(i+m,j,k) + p(i-m,j,k) - 2p(i,j,k)\right] + c_{ym}\left[p(i,j+m,k) + p(i,j-m,k) - 2p(i,j,k)\right] + c_{zm}\left[p(i,j,k+m) + p(i,j,k-m) - 2p(i,j,k)\right]$$

(Meng et al., arXiv 2020)

Experimental Setup

Computer	Hardware		Software
Summit	CPUs	2x IBM Power9	LLVM 9.0
	CPU cores	44 (22 per CPU)	
	Memory	512 GB	
	L3	10 MB (per two cores)	
	L2	512 KB (per two cores)	
	L1	32+32 KB	
	Device fabrication	14nm	
Cori	CPUs	2x Intel Xeon E5-2698v3	LLVM 10.0
	CPU cores	32 (16 per CPU)	
	Memory	128 GB	
	L3	40 MB (per socket)	
	L2	256 KB	
	L1	32+32 KB	
	Device fabrication	22nm	
SeaWulf	CPUs	2x Intel Xeon Gold 6148	LLVM 11.0 (git 3cd13c4)
	CPU cores	40 (20 per CPU)	
	Memory	192 GB	
	L3	28 MB (per socket)	
	L2	1024 KB	
	L1	32+32 KB	
	Device fabrication	14nm	