Lightweight Scheduling for Balancing the Tradeoff Between Load Balance and Locality

Vivek Kale[1] Simplice Donfack[3] Laura Grigori[2] William D. Gropp[1]

[1] University of Illinois at Urbana-Champaign [2]INRIA-Saclay [3]Oak Ridge National Laboratory

Maintaining Load Balance and Locality

Problem of Load Imbalance

- Load balance necessary to make efficient use of computational resources.
- Load imbalance comes from application (generally coarse-grained, persistent) *and* architecture (generally fine-grained, transient).

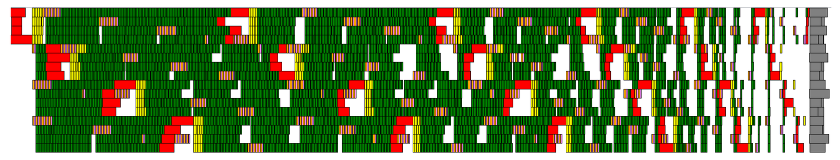
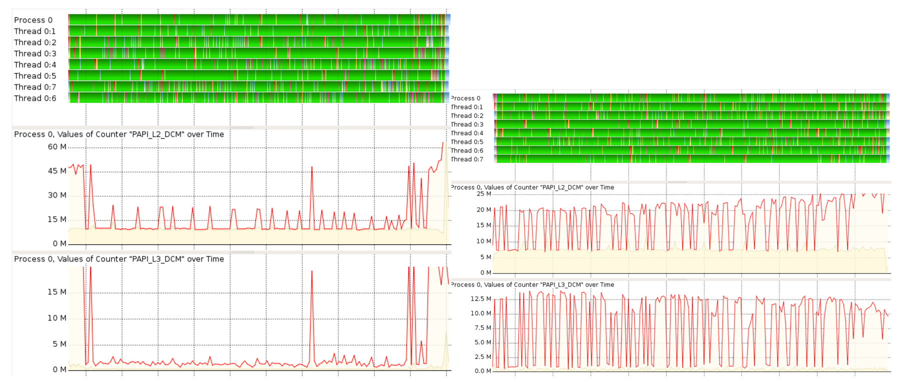


Figure 1: Communication-avoiding LU timeline with a static scheduling approach, with performance loss due to load imbalance shown by white spaces.

Problem with a Runtime-based Solution

- Runtime-based solution dynamic scheduling incurs overheads.
- Need to carefully balance for load balance and locality for application and architecture.



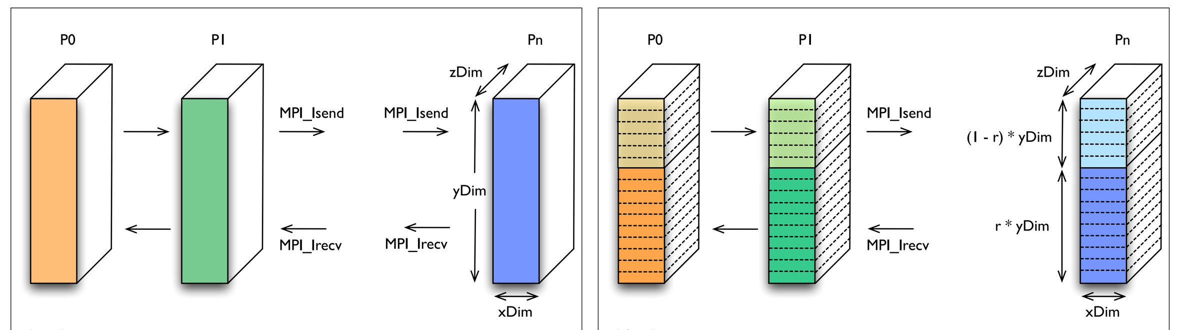
(a) Scheduling with Locality Tags: Reasonable cache locality, but limited load balance.

(b) Scheduling with Locality Tags and Work-stealing: Reasonable load balance, but low cache locality.

Figure 2: Timelines along with L2 and L3 cache misses.

Key Idea of Solution

1. Cores first do pre-assigned static iterations, and then do remaining iterations dynamically.
2. Ratio of static iterations to all iterations is the *static fraction* (denoted by r).
3. Tune static fraction to maintain both load balance and locality.



(a) Basic domain decomposition for 3D stencil.

(b) Mixed static/dynamic scheduling for 3D stencil.

Within-node Results and Analysis

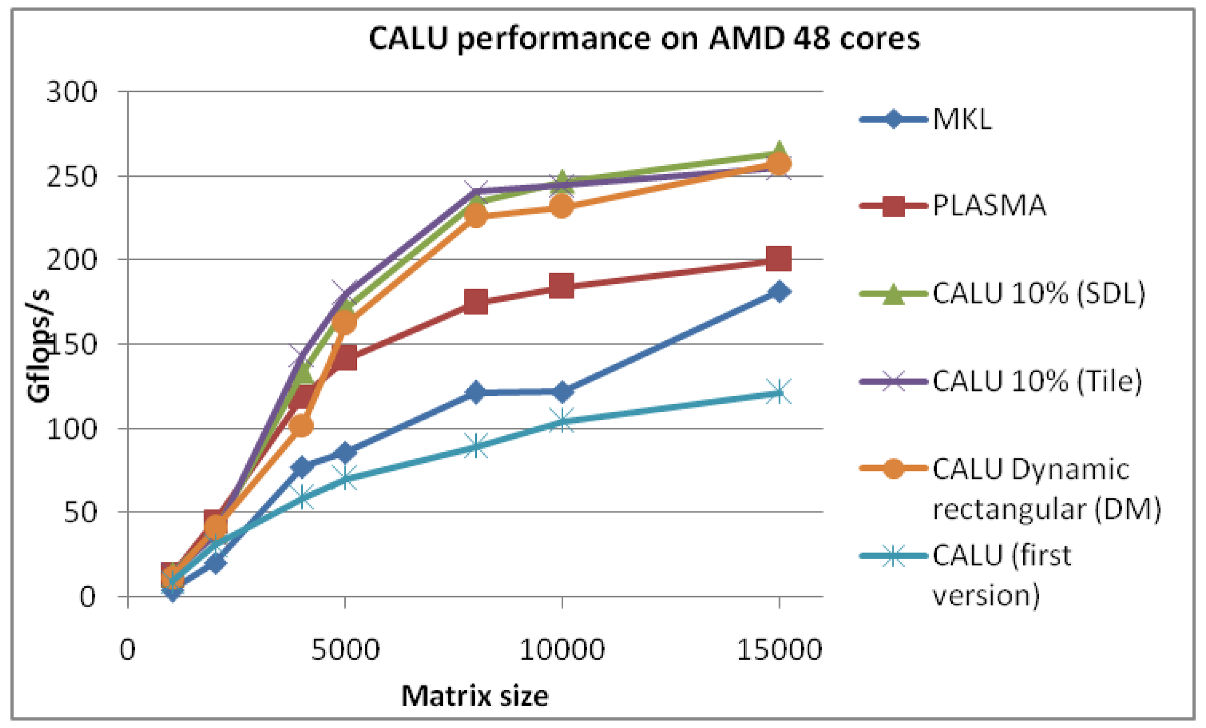
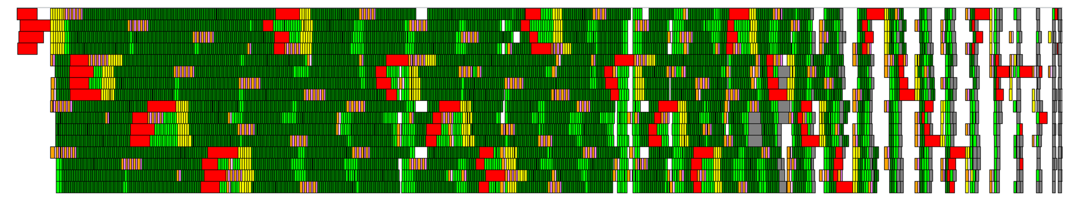


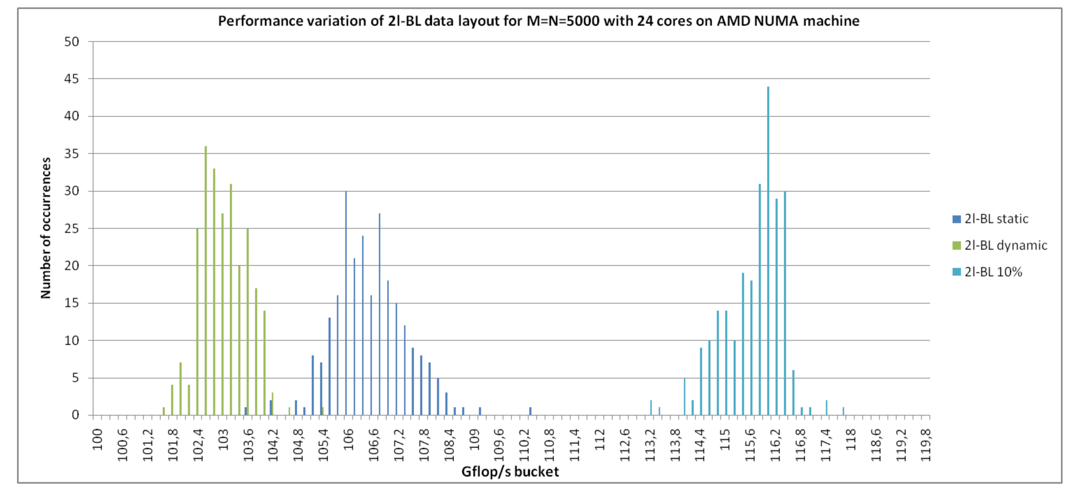
Figure 3: Performance of CALU using Mixed Static/dynamic Scheduling performs better compared to widely known numerical linear algebra libraries.

SP	BT	LU	FT	CG	MG
4.14%	5.42%	5.57%	5.31%	14.67%	9.48%

Table 1: Percent gains of mixed static/dynamic scheduling over OpenMP static scheduling for NAS benchmarks on Intel Westmere multi-core node



(a) Static/dynamic scheduling CALU Application Profile. Here, the performance loss due to load imbalance is reduced, seen through reduction in white in timeline.



(b) Performance variation of multiple executions of CALU is low with mixed static/dynamic scheduling, suggesting benefits to scalability.

Figure 4: Performance analysis of CALU on Intel Westmere 16-core cluster.

SP	BT	LU	FT	CG	MG
-1.09%	-1.05%	-1.62%	-1.59%	7.93%	5.04%

Table 2: Percent gains with NAS benchmarks on BG/Q multi-core node.

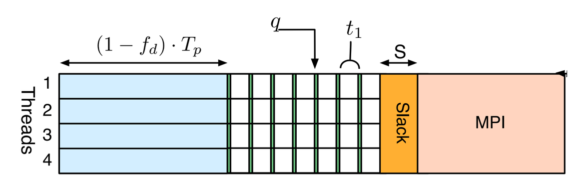
Implementation and Strategy Optimization

Scheduling Strategy Optimizations

- mixed static/dynamic(base)
- slack-conscious scheduling
- variable-task sizes
- weighted locality-sensitive
- constrained staggering
- → Each opens up new opportunities to balance the tradeoff between load balance and locality.

Methodology for Tuning Scheduler Parameters

- model-guided optimization
- runtime adjustment
- experimental auto-tuning
- → Each are important mechanisms to find the best tradeoff between load balance and locality.



Slack-conscious Strategy

$$f_d = \frac{p\delta}{N(t_1+q)}$$

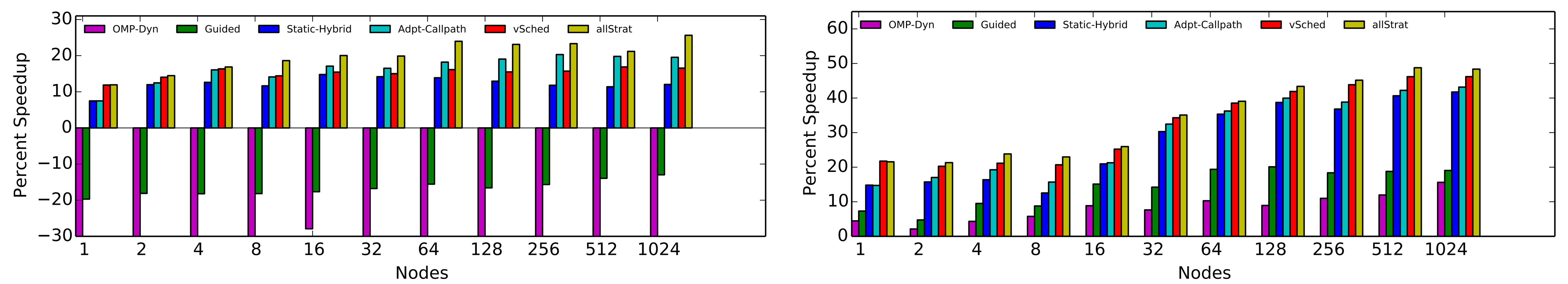
$$f_d' = f_d - \frac{S}{(p-1)\frac{Nt_1}{p} - Nq}$$

Model-guided Tuning

```
static LoopTimingRecord *record = NULL;
int startInd = 0;
int endInd = 0;
double fs;
fs = predict_static_fraction(&record);
#pragma omp parallel
{
  set_static_fraction(fs);
  startInd = 0; endInd = 0;
#pragma omp private(sum,k, startInd, endInd)
  FORALL_BEGIN(statdynstaggered, 1, lastrow-firstrow+2, startInd,
    endInd, omp_get_thread_num(), omp_get_num_threads(), startInd,
    endInd)
  for (j = startInd; j <= endInd; j++)
    {
      sum = 0.0;
      for (k = rowstr[j]; k < rowstr[j+1]; k++) {
        sum = sum + a[k]*p[colidx[k]];}
      w[j] = sum;
    }
}
end_timing(&record, n);
```

Figure 5: NAS CG code modification using our scheduling library.

Across-node Results and Scalability



(a) MPI+OpenMP reg. mesh CORAL SNAP code. Full strategy had 14 more lines of code (+15%) than original.

(b) MPI+OpenMP galaxy simulation Rebound code. Had 38 more lines of code(+4%).

Figure 6: Speedups for diff. sched. strategies for strong scaling on Intel Westmere 16-core cluster.

Results Summary

- At full-scale, reg. mesh obtains good gains of 25.52% over static sched. with slack-conscious sched. (*callsite_fd*), and has reasonable gains of 10.10% over static sched. with constrained staggered sched. (*vSched*).
- At full-scale, n-body gets limited gains of 7.56% over mixed static/dynamic (*Static-Hybrid*) with *callsite_fd*, but gets significant gains of 18.53% over mixed static/dynamic with *vSched*.
- The *full* scheduling, which combines *callsite_fd* and *vSched*, provides overall gains of 28.16% for reg. mesh, and 44.45% for n-body.

Conclusions and Future Work

- Important to tune the balance between load balance and locality.
- Using our approach, we achieve significant gains with a relatively low programmer effort, and perform competitively to guided sched.
- Additional gains can be achieved in *full* strategy, through more more intelligent tuning of the parameters of multiple schedulers.