Adaptive Loop Scheduling with Charm++ to Improve Performance of Scientific Applications

Vivek Kale[1] **Harshitha Menon**[2] **Karthik Senthil**[3]

[1] University of Southern California [2] Lawrence Livermore National Laboratory [3] University of Illinois at Urbana-Champaign

An Intelligent Runtime System for Clusters of SMPs

Challenges of using Charm++ on Multi-core

- An adaptive runtime system like Charm++ intelligently balances computational work periodically.
- Two issues exist when using a basic Charm++ scheduling scheme.
  1. Challenge of the cost of over-decomposition.
  2. Challenge and opportunity to exploit multi-core nodes to mitigate imbalance.
- We can address both challenges by (see Figure 1):
  1. Having Charm++'s load balancer assign Charm++ objects, i.e., chares, to nodes.
  2. Using loop parallelism to distribute work within a node.

Using Existing Intelligent Strategies

- The loop parallelism must handle dynamic imbalances while preserving locality.
- The mixed static/dynamic scheduling previously developed is a promising scheduling strategy for this purpose.
- A solution using only within-node loop scheduling won't handle inter-node load balance that's significant in many applications.

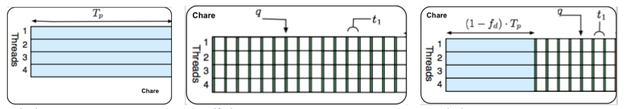
→ Need to modify load balancer so that it uses loop scheduling and is adjusted to reduce overhead of load balancing.

Key Idea of Our Solution

1. Modify Charm++ RTS to assign chares to core 0 of each node only.
2. Reduce the number of chares per PE.
3. Adjust parameters of within-node loop scheduler,e.g., vary static fraction, based on parameters of Charm++.
4. Tune Charm++ RTS parameters to work with adjustments of within-node loop scheduling.

Figure 1: Timelines using Charm++-only for default (left) and a mode where the number of chares per node is reduced (right).

Figure 2: Timelines for execution of a code without (left) and with (right) Charm++ load balancing.



(a) Static Scheduling. (b) Dynamic Scheduling. (c) Mixed Scheduling.

Figure 3: Mixed static/dynamic scheduling within a chare, i.e., a Charm++ object.

Baseline Results and Analysis

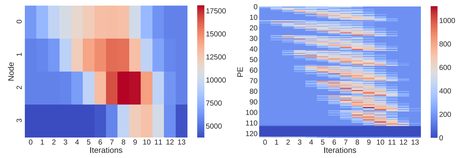


Figure 4: Load imbalances across nodes (left) and across cores (right) when no load balancing strategy used.

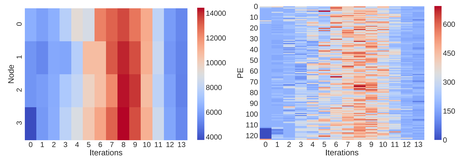


Figure 5: Load imbalances across nodes (left) and across cores (right) when the greedy load balancing strategy is used.

Existence of Within-node Load Imbalance

- Using no load balancing, node 2 is heavily overloaded for iterations 8 and 9. Hence, we need load balancing to distribute the load across nodes.
- Inter-node load balancing using GreedyLB balances load across nodes well.
- Balancing load across nodes using GreedyLB still leaves load imbalance in the cores within a node.

Figure 6: Case Study on AMR: Even though the computational work is well-distributed, the serial portions of the code running on core 0 degrade performance.

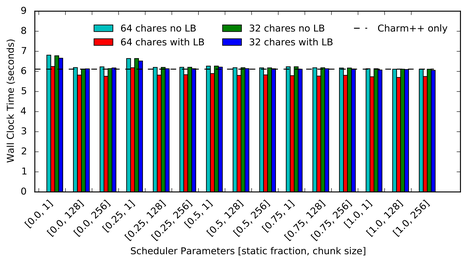


Figure 7: Within node results for Particle-in-Cell code using different loop scheduling and load balancing strategies.

Strategies and Tuning

t_1	Duration of a loop iteration
T_p	Total execution time of a threaded computation region consisting of N loop iterations on p cores
δ	Expected cost of load imbalance on a node to the application
$load_i$	Load on the i^{th} core of a node on an arbitrary timestep

Table 1: Terms in implementation

Scheduling Strategies

- Mixed static/dynamic
- Weighted dynamic
- Staggered

Load Balancing Strategies

- Periodic
- Hierarchical

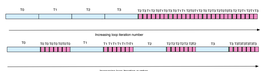


Figure 8: Mixed static/dynamic scheduling using an optimization of iteration staggering

Parameters of Scheduler

1. Static fraction.
2. Chunk size.
3. Distribution of chunk sizes in queue.
4. Cores per steal queue.

Parameters of Load Balancer

1. Chares per PE.
2. Frequency of load balancing step.

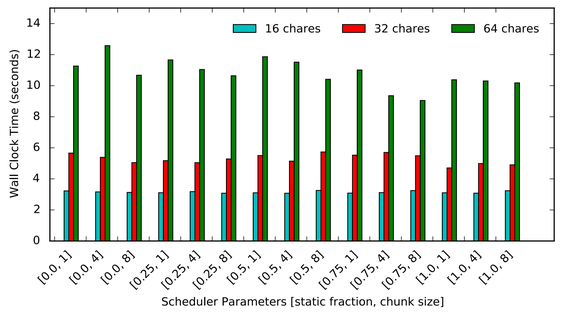
Methodology for Tuning Parameters

- The parameters of the load balancer and of the scheduler can't be tuned independently.
- We tune the parameters of the scheduler and load balancer together.

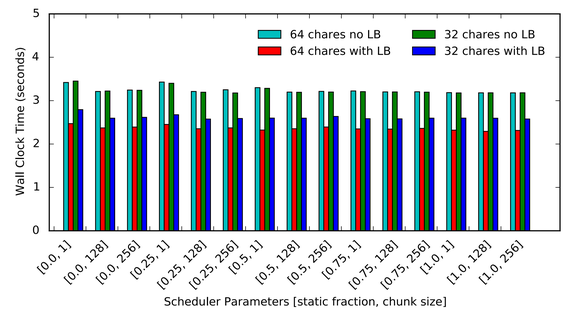
```
#include "charm++.h"
#include "CkLoopAPI.h"
void doDotProduct()
{
  float r = 0.0; int numberOfChunks= (probSize/numElems)/chunkSize;
  float* params[2]; params[0] = a; params[1] = b;
  CkLoop_ParallelizeHybrid(lh, dotP_chunked, 2, params, numberOfChunks
    , 0, (probSize/numElems), 1, &r, CKLOOP_FLOAT_SUM);
  CkCallback cb(CkReductionTarget(Main, printResult), mainProxy);
  contribute(sizeof(float), &r, CkReduction::sum_float, cb);
}
extern "C" void dotP_chunked(int start, int end, void* result, int
  numParams, void* params)
{
  float** z = (float**) params; float* v1 = z[0];  float* v2 = z[1];
  float x = 0.0;for(int i=start; i<end; i++) x += a[i]*b[i]; * ((
    double**)result) = x;
}
class rank0BlockMap : public CkArrayMap
{
public:
  rank0BlockMap(void) {}
  rank0BlockMap(CkMigrateMessage *m){}
  int registerArray(CkArrayIndex& numElements,CkArrayID aid){return
    0;}
  int procNum(int /*arrayHdl*/, const CkArrayIndex &idx) {
    int elem=*(int *)idx.data(); int charesPerNode = numElemsPerNode;
    int nodeNum = (elem/(charesPerNode)); int numPEsPerNode = CkNumPes
      ()/CkNumNodes();
    int penum = nodeNum*numPEsPerNode; return penum;
  }
};
```

Figure 9: Dot product code's modification using Charm++ with CkLoop.

Across-node Results and Evaluation



(a) Dot Product on 4 nodes of Stampede.



(b) Particle-in-Cell on 4 nodes of Blue Waters.

Figure 10: Execution times of scientific applications on multi-core clusters.

Results Summary

- For dot product using 64 chares, a static fraction of 75% with a chunk size of 8 gives the best performance, showing utility of adaptive loop scheduling.
- PIC using modified inter-node load balancing with adaptive loop scheduling is 19.13% faster than PIC using adaptive scheduling without load balancing.
- Synergistic performance improvements using combination of inter-node and intra-node load balancing.

Conclusions and Future Work

- Need sophisticated loop scheduling in Charm++.
- Described a technique and implementation to improve performance of applications.
- Using our technique and implementation, we demonstrate performance improvement of 17.2%.
- For future work, we'll explore other adjustments to parameters of the Charm++ RTS to facilitate for loop scheduling.