Optimized Distributed Work-Stealing

Vivek Kumar\textsuperscript{1}, Karthik Murthy\textsuperscript{1}, Vivek Sarkar\textsuperscript{1}, Yili Zheng\textsuperscript{2}

\textsuperscript{1} Rice University
\textsuperscript{2} Lawrence Berkeley National Laboratory
Multicore Nodes in Supercomputers

Cores/Socket System Share in Top500

Graph plotted using the data obtained from https://www.top500.org/statistics/list/

November 2006

1 (45%)

2 (55%)

June 2016

8 (21.6%)

10 (15%)

6 (13.2%)

16 (10%)

12 (30%)

Others (10%)
Productivity and Performance Challenge

• Productivity
  – Several existing APIs for scientific computing
  – Hard to parallelize complex irregular computations using existing APIs
    • Ideal candidate for runtime based global load-balancing

• Performance on multicore nodes
  – Using a process per core (e.g., MPI everywhere) on a node not scalable
  – Hybrid programming using thread pool per node
    • How to design a high performance implementation of global load-balancing
Contributions

✔ Library-based API in a PGAS library to express irregular computations
  C++11 lambda function based API that provides serial elision

✔ Novel implementation of distributed work-stealing
  That introduces a new victim selection policy that avoid all inter-node failed steals

✔ Detailed performance study
  That demonstrates the benefit using scaling irregular applications up to 12k cores of Edison supercomputer

✔ Results
  That shows that our approach delivers performance benefits up to 7%
Motivating Analysis

Load Balancing using Work-Stealing

- Thread pool (intra-node) based implementations perform stealing using low overhead CAS operations
Motivating Analysis

Distributed Work-Stealing

- **Inter-node** steals are much costlier than **intra-node** steals
Failed Steal Attempts

• Thief fails to steal a task from victim

**Inter-node** failed steals are more costly than **intra-node** steals

Chances to fail with same victim multiple times

**Intra-node** failed steals

**Inter-node** failed steals
Our Approach

• Use HabaneroUPC++ PGAS library for multicore cluster [Kumar et al., PGAS 2014]
  – Several asynchronous tasking APIs
• Provide a programming model to express irregular computation
• Implement a high performance distributed work-stealing runtime that completely removes all inter-node failed steal attempts
HabaneroUPC++ Programming Model

```cpp
asyncAny ( [] {
    irregular_computation();
}); //distributed work-stealing
```

- C++11 lambda-function based API
- Provides **serial elision** and improves productivity
Distributed Work-Stealing Runtime

• Two different implementations in HabaneroUPC++
  • BaselineWS
    – Uses prior work + some optimizations
  • SuccessOnlyWS
    – Extends BaselineWS by using a novel victim selection policy that completely removes all inter-node failed steals
Step-1: Failed steal at intra-node level

Step-2: Local worker request the leader for inter-node steal

Step-3: Leader finds a victim that has sufficient number of tasks (RDMA)

Step-4: Lock and wait for tasks from victim

Step-5: Remote leader attempts to steal tasks from its local thread-pool

Step-6: Remote victim can send tasks, else it's a inter-node failed steal
Optimized Distributed Work-Stealing | Kumar et al. | IA³ 2016

SuccessOnlyWS in HabaneroUPC++

Step-1: Failed steal at intra-node level

Step-2: Local worker request the leader for inter-node steal

Step-3: Leader finds a victim that has sufficient number of tasks (RDMA)

Step-4: Asynchronous task request

Step-5: Remote leader attempts to steal tasks from its local thread-pool

Step-6: Repeat Step-3 and Step-4

Step-7: One or more remote victim will send tasks. Break out of Step-5 (also if application terminates)
Methodology

• Benchmarks
  – Two UTS trees T1WL and T3WL
  – NQueens
• Computing infrastructure
  – Edison supercomputer at NERSC
    • 2x12 cores per node
Experimental Evaluation

Results

Higher inter-node failed steals in BaselineWS => Better performance in SuccessOnlyWS

(a) Total inter-node failed steals in BaselineWS

(b) SuccessOnlyWS speedup over BaselineWS

Note: More results are available in the paper
Summary and Conclusion

• Inter-node steals are costlier than intra-node steals
• Failed inter-node steals could hamper performance
• C++11 lambda function based API to in HabaneroUPC++ to express complex irregular computation that can participate in distributed work-stealing
• A novel implementation of distributed work-stealing runtime in HabaneroUPC++ PGAS library that completely removes all inter-node failed steals
• Our novel runtime delivers performance benefits up to 7%
Backup Slides
Existing Techniques for Inter-node Stealing

- Thread pool based hybrid runtimes [Lifflander et. al., HPDC’12, Paudel et. al., ICPP’13]
- Communication worker maintain ready queue of tasks even before a remote request arrives [Paudel et. al., ICPP’13]
- Load-aware steal attempts to reduce chances of failure [Dinan et. al., ICPP’08]
- First try random victims and on failing contact set of victims (lifelines) that promises to send tasks whenever they have it ready [Saraswat et. al., PPoPP’11]
procedure Steal_AsyncAny
while (global termination is not detected)
    V = get a random remote rank
    if (V has declared task availability in PGAS space) // RDMA
        if (I did not try to steal from V)
            queue my rank at V
        if TryLock (V) is success
            save my rank at V
            wait until V send tasks or decline
            Unlock (V)
    break from while loop if I just received asyncAny tasks
if I receive asyncAny tasks from any victim
    forget that I contacted this victim
    reset my task receiving status
Inter-node Task Transfer from Victim

procedure Send_AsyncAny

while (there are pending inter-node steal requests)

    T = get rank of one of the queued remote thief
    steal tasks from my local workers and send to T
    forget that T contacted me
    break out of the while loop if local steal failed

T = get rank of the only waiting remote thief
steal tasks from local workers and send to T
declare that now I don’t have any waiting remote thief
publish in PGAS space asyncAny count at my place