Argobots and its Application to Charm++

Sangmin Seo
Assistant Computer Scientist
Argonne National Laboratory
sseo@anl.gov

April 19, 2016

Charm++ Workshop 2016
Argo Concurrency Team

- Argonne National Laboratory (ANL)
  - Pavan Balaji (co-lead)
  - Sangmin Seo
  - Abdelhalim Amer
  - Marc Snir
  - Pete Beckman (PI)
- University of Illinois at Urbana-Champaign (UIUC)
  - Laxmikant Kale (co-lead)
  - Prateek Jindal
  - Jonathan Lifflander
- University of Tennessee, Knoxville (UTK)
  - George Bosilca
  - Thomas Herault
  - Damien Genet
- Pacific Northwest National Laboratory (PNNL)
  - Sriram Krishnamoorthy

Past Team Members:
- Cyril Bordage (UIUC)
- Esteban Meneses (University of Pittsburgh)
- Huiwei Lu (ANL)
- Yanhua Sun (UIUC)
Massive On-node Parallelism

- The number of cores is increasing
- Massive on-node parallelism is inevitable
- Existing solutions do not effectively deal with such parallelism with respect to on-node threading/tasking systems or with respect to off-node communication in the presence of such tasks/threads
- How to exploit?

Core-level Parallelism
**Shortcomings today? Pthreads (1/2)**

**Nesting**

```c
int in[1000][1000], out[1000][1000];

#pragma omp parallel for
for (i = 0; i < 1000; i++) {
    Petsc_voodoo(i);
}

Petsc_voodoo(int x)
{
    #pragma omp parallel for
    for (j = 0; j < 1000; j++)
        out[x][j] = cosine(in[x][j]);
}
```

Execution time for 36 threads in the outer loop

Why is traditional OpenMP’s performance so bad? The compiler cannot analyze Petsc_voodoo to know whether the function might ever block or yield, so it has to assume that it might. Therefore a stack is needed to facilitate it. Creating additional Pthreads for each nesting is the simplest way to achieve this.
Shortcomings today? Pthreads (2/2)

Tasks of application mapped to a group of Pthreads

How about these communications? Wait or context switch?

Work units intermixed with blocking calls (such as communication calls) can cause idle cores

Need lightweight mechanisms to switch tasks!
Outline

• Background
• Argobots
• Charm++ with Argobots
• Other Programming Models
• Summary
User-Level Threads (ULTs)

• What is user-level thread (ULT)?
  – Provides thread semantics in user space
  – Execution model: **cooperative timesharing**
    • More than one ULT can be mapped to a single kernel thread
    • ULTs on the same OS thread do not execute concurrently

• Where to use?
  – To better overlap computation and communication/IO
  – To exploit fine-grained task parallelism
Pthreads vs. ULTs

- Average time for creating and joining one thread
  - pthread: 6.6us - 21.2us (avg. 34,953 cycles)
  - ULT (Argobots): 78ns - 130ns (avg. 191 cycles)
  - ULT is 64x - 233x faster than Pthread

- How fast is ULT?
  - L1$ access: 1.112ns, L2$ access: 5.648ns, memory access: 18.4ns
  - Context switch (2 processes): 1.64us

* measured using LMbench3
Growing Interests in ULTs

- ULT and task libraries
  - Converse threads, Qthreads, MassiveThreads, Nanos++, Maestro, GnuPth, StackThreads/MP, Protothreads, Capriccio, StateThreads, TiNy-threads, etc.

- OS supports
  - Windows fibers, Solaris threads

- Language and programming models
  - Cilk, OpenMP task, C++11 task, C++17 coroutine proposal, Stackless Python, Go coroutines, etc.

- Pros
  - *Easy to use with Pthreads-like interface*

- Cons
  - *Runtime tries to do something smart (e.g., work-stealing)*
  - *This may conflict with the characteristics and demands of applications*
Argobots

A low-level lightweight threading and tasking framework
(http://collab.cels.anl.gov/display/argobots/)

Overview
• Separation of mechanisms and policies
• Massive parallelism
  – Exec. Streams guarantee progress
  – Work Units execute to completion
    • User-level threads (ULTs) vs. Tasklet
• Clearly defined memory semantics
  – Consistency domains
    • Provide Eventual Consistency
  – Software can manage consistency

Argobots Innovations
• Enabling technology, but not a policy maker
  – High-level languages/libraries such as OpenMP, Charm++ have more information about the user application (data locality, dependencies)
• Explicit model:
  – Enables dynamism, but always managed by high-level systems

* Team members: Sangmin Seo, Abdelhalim Amer, Pavan Balaji (ANL), Laxmikant Kale, Prateek Jindal (UIUC)
Argobots Execution Model

• **Execution Streams (ES)**
  – Sequential instruction stream
    • Can consist of one or more work units
  – Mapped efficiently to a hardware resource
  – Implicitly managed progress semantics
    • One blocked ES cannot block other ESs

• **User-level Threads (ULTs)**
  – Independent execution units in user space
  – Associated with an ES when running
  – Yieldable and migratable
  – Can make blocking calls

• **Tasklets**
  – Atomic units of work
  – Asynchronous completion via notifications
  – Not yieldable, migratable before execution
  – Cannot make blocking calls

• **Scheduler**
  – Stackable scheduler with pluggable strategies

• **Synchronization primitives**
  – Mutex, condition variable, barrier, future

• **Events**
  – Communication triggers
**Explicit Mapping ULT/Tasklet to ES**

- The user needs to map work units to ESs
- No smart scheduling, no work-stealing unless the user wants to use

![Diagram showing mapping of work units to ESs]

**Benefits**

- Allow locality optimization
  - Execute work units on the same ES
- No expensive lock is needed between ULTs on the same ES
  - They do not run concurrently
  - A flag is enough
Stackable Scheduler with Pluggable Strategies

- Associated with an ES
- Can handle ULTs and tasklets
- *Can handle schedulers*
  - Allows to stack schedulers hierarchically
- Can handle asynchronous events
- *Users can write schedulers*
  - Provides **mechanisms**, not policies
  - Replace the default scheduler
    - E.g., FIFO, LIFO, Priority Queue, etc.
- ULT can explicitly **yield to** another ULT
  - Avoid scheduler overhead

```plaintext
yield()
yield_to(target)
```
Performance: Create/Join Time

- Ideal scalability
  - If the ULT runtime is perfectly scalable, the time should be the same regardless of the number of ESs
Charm++ with Argobots

Jonathan Lifflander, Prateek Jindal, Yanhua Sun
Laxmikant Kale

University of Illinois at Urbana-Champaign (UIUC)
Charm++ with Argobots

• Goals
  – Test the completeness and performance of Argobots with Charm++ programming model
  – Take advantage of Argobots features (tasklets, stackable schedulers, etc.) without modifying application codes
  – For Charm++ applications, interoperate with applications written in other models (MPI, Cilk, etc.)

* Team members: Laxmikant Kale, Jonathan Lifflander, Prateek Jindal (UIUC)
Replacing the Converse Runtime with Argobots

- **Converse**
  - The active messaging layer in Charm++
- **Approaches**
  - Each Charm++ Pthread inside a node (including the communication thread) is implemented as an Argobots ES
    - Create an ES for every Converse instance
  - A custom Argobots scheduler is created instead of using the Converse scheduler
  - Converse messages are enqueued into Argobots pools as tasklets
  - Converse threads (CthThread) are implemented on top of Argobots ULTs, with conditional variables to implement suspend/resume

- **Only 180 lines of code had to be changed!**
LeanMD Performance: Runtime Comparison

- Evaluation machine
  - 2 x Intel Xeon E5-2699 v3 (2.30GHz): 36 cores (72 threads)
- LeanMD simulation
  - A total of 20 steps on a cell array of dimensions 7x7x7
  - 1-away XYZ configuration and 1000 atoms per cell

Achieved comparable performance although it is a very simple implementation
LeanMD Performance: Manual Implementation

- Manual implementation of LeanMD using the Argobots
  - Exploited both ULTs and tasklets
    - A ULT for managing a cell and a tasklet for managing the interaction between cells
    - Used futures for the waiting mechanism
    - Work stealing between pools
- Better performance of our manual implementation implies that Charm++ with Argobots could be improved

![Graph showing execution time comparison between Charm++ and Argobots]
Argobots: Interfaces for Shrink/Expand Events

1. [Argobots] Connect to NRM using a socket on ABT_init()
2. [Runtimes/applications] Register callback functions for shrink/expand events
3. [Runtimes/applications] Deregister callback functions when they terminate
4. [Argobots] Disconnect from NRM on ABT_finalize()
Argobots: Shrink/Expand Event Handling

- Shrinking

1. ES$_1$ picks an event, which requests ES$_2$ to be stopped
2. Ask the runtime using callbacks whether ES$_2$ can be stopped
3. If OK, mark ES$_2$ to need to stop so when the scheduler on ES$_2$ checks events, it can be stopped
4. Notify the runtime that ES$_2$ will be stopped
5. Create a ULT on ES$_0$
6. When the scheduler on ES$_2$ stops, ES$_2$ is terminated
7. After ES$_2$ is terminated, the ULT frees ES$_2$ and sends a response to NRM
Argobots: Shrink/Expand Event Handling

• Expanding

1. ES₀ picks an event, which requests to create an ES₂
2. Ask the runtime using callbacks whether it can create ES₂
3. If OK, invoke a callback function so the runtime creates ES₂
4. Create ES₂
5. Send a response to NRM
Charm++ with Argobots: Implementation

- Shrink/Expand Implementation
  - Charm++ maintains a set of pools for each scheduler, mapped to an ES
  - Ranks in Charm++ are virtualized by saving the mapping of pool to an ES
  - When an ES is removed, the associated pools are put into a global list
    - To maintain correctness in Charm++, the rank of any tasks/threads in the global list are derived from the pool (ranks are virtualized)
  - Shrink
    - Other ESs execute work units from orphaned pools with some added synchronization
  - Expand
    - A new ES is created and takes over a set of orphaned pools
LeanMD Results of Shrinking/Expanding ESs

(a) Execution Streams

(b) Application Performance

(c) Performance per Watt
Argobots Ecosystem

MPI+Argobots

Applications
- Charm++
- Argobots runtime

Communication libraries

OmpSs

CilkBots

ES1

Argobots

Sched

ESn

Global Thread Team

Fused ULT N
Fused ULT 2
Fused ULT 1
RWS ULT

Argobots ES

Cilk “Worker”

PaRSEC

MPI

OpenMP

External Connections

GridFTP, Kokkos, RAJA, ROSE, TASCEL, XMP, etc.

External Connections

Origin

RPC proc

Target

Mercury RPC

Charm++ Workshop 2016
Summary

• Massive on-node parallelism is inevitable
  – Need runtime systems utilizing such parallelism

• Argobots
  – A lightweight low-level threading/tasking framework
  – Provides efficient mechanisms, not policies, to users (library developers or compilers)
  • They can build their own solutions

• Charm++ with Argobots
  – Implemented by replacing the Converse runtime with Argobots
  – Achieved comparable performance on LeanMD
  – Incorporated the shrinking/expanding of ESs in Argobots in order to respond to the external events (e.g., power capping)
Try Argobots

• git repository
  • http://git.mcs.anl.gov/argo/argobots.git/

• Documentation
  – Wiki
    • https://collab.cels.anl.gov/display/ARGOBOTS/
  – Doxygen
    • http://www.mcs.anl.gov/~sseo/public/argobots/
Funding Acknowledgments

**Funding Grant Providers**

- Argonne National Laboratory
- U.S. Department of Energy, Office of Science
- NSF
- National Nuclear Security Administration
- Sandia National Laboratories
- Microsoft
- NCSA

**Infrastructure Providers**

- Argonne National Laboratory
- U.S. Department of Energy, Office of Science
- NSF
- National Nuclear Security Administration
- Sandia National Laboratories
- Microsoft
- NCSA
- Intel
- IBM
Programming Models and Runtime Systems Group

**Group Lead**
- Pavan Balaji (computer scientist and group lead)

**Current Staff Members**
- Abdelhalim Amer (postdoc)
- Yanfei Guo (postdoc)
- Rob Latham (developer)
- Lena Oden (postdoc)
- Ken Raffenetti (developer)
- Sangmin Seo (assistant computer scientist)
- Min Si (postdoc)
- Min Tian (visiting scholar)

**Past Staff Members**
- Antonio Pena (postdoc)
- Wesley Bland (postdoc)
- Darius T. Buntinas (developer)
- James S. Dinan (postdoc)
- David J. Goodell (developer)
- Huiwei Lu (postdoc)
- Yanjie Wei (visiting scholar)
- Yuqing Xiong (visiting scholar)
- Jian Yu (visiting scholar)
- Junchao Zhang (postdoc)
- Xiaomin Zhu (visiting scholar)

**Current and Recent Students**
- Ashwin Aji (Ph.D.)
- Abdelhalim Amer (Ph.D.)
- Md. Humayun Arafat (Ph.D.)
- Alex Brooks (Ph.D.)
- Adrian Castello (Ph.D.)
- Dazhao Cheng (Ph.D.)
- James S. Dinan (Ph.D.)
- Piotr Fidkowski (Ph.D.)
- Priyanka Ghosh (Ph.D.)
- Sayan Ghosh (Ph.D.)
- Ralf Gunter (B.S.)
- Jichi Guo (Ph.D.)
- Yanfei Guo (Ph.D.)
- Marius Horga (M.S.)
- John Jenkins (Ph.D.)
- Feng Ji (Ph.D.)
- Ping Lai (Ph.D.)
- Palden Lama (Ph.D.)
- Yan Li (Ph.D.)
- Huiwei Lu (Ph.D.)
- Jintao Meng (Ph.D.)
- Ganesh Narayanaswamy (M.S.)
- Qingpeng Niu (Ph.D.)
- Ziaul Haque Olive (Ph.D.)
- David Ozog (Ph.D.)
- Renbo Pang (Ph.D.)
- Sreeram Potluri (Ph.D.)
- Li Rao (M.S.)
- Gopal Santhanaraman (Ph.D.)
- Thomas Scogland (Ph.D.)
- Min Si (Ph.D.)
- Brian Skjerven (Ph.D.)
- Rajesh Sudarsan (Ph.D.)
- Lukasz Wesolowski (Ph.D.)
- Shucai Xiao (Ph.D.)
- Chaoran Yang (Ph.D.)
- Boyu Zhang (Ph.D.)
- Xiuxia Zhang (Ph.D.)
- Xin Zhao (Ph.D.)

**Advisory Board**
- Pete Beckman (senior scientist)
- Rusty Lusk (retired, STA)
- Marc Snir (division director)
- Rajeev Thakur (deputy director)
Q&A

- Thank you for your attention!

Questions?