Welcome to the 2017 Charm++ Workshop!

Laxmikant (Sanjay) Kale

http://charm.cs.illinois.edu
Parallel Programming Laboratory
Department of Computer Science
University of Illinois at Urbana Champaign
A bit of history

• This is the 15$^{th}$ workshop in a series that began in 2001
Charm++ Workshops

- Upcoming Workshop 2017
- Workshop 2016
- Workshop 2015
- Workshop 2014
- Workshop 2013
- Workshop 2012
- Workshop 2011
- Workshop 2010
- Workshop 2009
- Workshop 2008
- Workshop 2007
- Workshop 2005
- Workshop 2004
- Workshop 2003
- Workshop 2002
A Reflection on the History

• Charm++, the name, is from 1993
• Most of the foundational concepts : by 2002
• So, what does this long period of 15 years signify?
• Maybe I was too slow
• But I prefer the interpretation:
  – We have been enhancing and adding features based on large-scale application development.
    • A long co-design cycle
  – The research agenda opened up by the foundational concepts is vast
  – Although the foundations were done in 2002, the fleshing out of adaptive runtime capabilities is where many intellectual challenges, and engineering work, lay.
What is Charm++?

• Charm++ is a generalized approach to writing parallel programs
  – An alternative to the likes of MPI, UPC, GA etc.
  – But not to sequential languages such as C, C++, Fortran

• Represents:
  – The style of writing parallel programs
  – The runtime system
  – And the entire ecosystem that surrounds it

• Three design principles:
  – Overdecomposition, Migratability, Asynchrony
Overdecomposition

• Decompose the work units & data units into many more pieces than execution units
  – Cores/Nodes/..

• Not so hard: we do decomposition anyway
Migratability

• Allow these work and data units to be migratable at runtime
  – i.e. the programmer or runtime, can move them

• Consequences for the app-developer
  – Communication must now be addressed to logical units with global names, not to physical processors
  – But this is a good thing

• Consequences for RTS
  – Must keep track of where each unit is
  – Naming and location management
Asynchrony: Message-Driven Execution

• With over decomposition and Migratibility:
  – You have multiple units on each processor
  – They address each other via logical names

• Need for scheduling:
  – What sequence should the work units execute in?
  – One answer: let the programmer sequence them
    • Seen in current codes, e.g. some AMR frameworks
  – Message-driven execution:
    • Let the work-unit that happens to have data (“message”) available for it execute next
    • Let the RTS select among ready work units
    • Programmer should not specify what executes next, but can influence it via priorities
Realization of this model in Charm++

- Overdecomposed entities: chares
  - Chares are C++ objects
  - With methods designated as “entry” methods
    - Which can be invoked asynchronously by remote chares
  - Chares are organized into indexed collections
    - Each collection may have its own indexing scheme
      - 1D, ..7D
      - Sparse
      - Bitvector or string as an index
  - Chares communicate via asynchronous method invocations
    - A[i].foo(....); A is the name of a collection, i is the index of the particular chare.
Message-driven Execution

A[23].foo(…)

Processor 0
Scheduler
Message Queue

Processor 1
Scheduler
Message Queue

2017 CHARM++ WORKSHOP
Processor 0
Scheduler
Message Queue

Processor 1
Scheduler
Message Queue

Processor 2
Scheduler
Message Queue

Processor 3
Scheduler
Message Queue
The Adaptive RTS can:
- Dynamically balance loads
- Optimize communication:
  - Spread over time, async collectives
- Automatic latency tolerance
- Prefetch data with almost perfect predictability
# Some Production Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Domain</th>
<th>Previous parallelization</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMD</td>
<td>Classical MD</td>
<td>PVM</td>
<td>500k</td>
</tr>
<tr>
<td>ChaNGa</td>
<td>N-body gravity &amp; SPH</td>
<td>MPI</td>
<td>500k</td>
</tr>
<tr>
<td>EpiSimdemics</td>
<td>Agent-based epidemiology</td>
<td>MPI</td>
<td>500k</td>
</tr>
<tr>
<td>OpenAtom</td>
<td>Electronic Structure</td>
<td>MPI</td>
<td>128k</td>
</tr>
<tr>
<td>Spectre</td>
<td>Relativistic MHD</td>
<td></td>
<td>100k</td>
</tr>
<tr>
<td>FreeON/SpAMM</td>
<td>Quantum Chemistry</td>
<td>OpenMP</td>
<td>50k</td>
</tr>
<tr>
<td>Enzo-P/Cello</td>
<td>Astrophysics/Cosmology</td>
<td>MPI</td>
<td>32k</td>
</tr>
<tr>
<td>ROSS</td>
<td>PDES</td>
<td>MPI</td>
<td>16k</td>
</tr>
<tr>
<td>SDG</td>
<td>Elastodynamic fracture</td>
<td></td>
<td>10k</td>
</tr>
<tr>
<td>ADHydro</td>
<td>Systems Hydrology</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>Disney ClothSim</td>
<td>Textile &amp; rigid body dynamics</td>
<td>TBB</td>
<td>768</td>
</tr>
<tr>
<td>Particle Tracking</td>
<td>Velocimetry reconstruction</td>
<td></td>
<td>512</td>
</tr>
<tr>
<td>JetAlloc</td>
<td>Stochastic MIP optimization</td>
<td></td>
<td>480</td>
</tr>
</tbody>
</table>
Intelligent, introspective, Adaptive Runtime Systems, developed for handling application’s dynamic variability, already have features that can deal with challenges posed by exascale hardware
Relevant capabilities for Exascale

• Load balancing
• Data-driven execution in support of task-based models
• Resilience
  – multiple approaches: in-memory checkpoint, leveraging NVM, message-logging for low MTBF
  – all leveraging object-based overdecomposition
• Power/Thermal optimizations
• Shrink/Expand sets of processors allocated during execution
• Adaptivity-aware resource management for whole-machine optimizations
IEEE Computer highlights Charm++ energy efficient runtime
Interaction Between the Runtime System and the Resource Manager

- Allows dynamic interaction between the system resource manager or scheduler and the job runtime system
- Meets system-level constraints such as power caps and hardware configurations
- Achieves the objectives of both datacenter users and system administrators
Charm++ interoperates with MPI

So, you can write one module in Charm++, while keeping the rest in MPI
Integration of Loop Parallelism

• Used for transient load balancing within a node
• Mechanisms:
  – Charm++’s old CkLoop construct
  – New integration with OpenMP (gomp, and now llvm)
  – BSC’s OMPSS integration is orthogonal
  – Other new OpenMP schedulers
• RTS splits a loop into Charm++ messages
  – Pushed into each local work stealing queue
    • where idle threads within the same node can steal tasks
for (i = 0; i < n; i++) {
    ...
}
Recent Developments: Charmworks, Inc.

• Charm++ is now a commercially supported system
  – Charmworks, Inc.
  – Supported by DoE SBIR and small set of initial customers
• Non profit use (academia, US Govt. Labs..) remains free
• We are bringing improvements made by Charmworks into the University version (no forking of code so far)
• Specific improvements have included:
  – Better handling of errors
  – Robustness and ease of use improvements
  – Production versions of research capabilities
• A new project at Charmworks for support and improvements to Adaptive MPI (AMPI)
Upcoming Challenges and Opportunities

• Fatter nodes
• Improved global load balancing support in presence of GPGPUs
• Complex memory hierarchies (e.g. HBM)
  – I think we are well-equipped for that, with prefetch
• Fine-grained messaging and lots of tiny chares:
  – Graph algorithms, some solvers, DES, ..
• Subscale-simulations, multiple simulations
• In-situ analytics
• Funding!
A glance at the Workshop

- **Keynotes:** Michael Norman, Rajeev Thakur
- **PPL taks:**
  - Capabilities: load balancing*, heterogenity, DES
  - Algorithms: sorting, connected components
- **Languages:** DARMA, Green-Marl, HPX (non-charm)
- **Applications:**
  - NAMD, ChaNGA, OpenAtom, multi-level summation
  - TaBaSCo (LANL, proxy app),
  - Quinoa (LANL, Adaptive CFD)
  - SpECTRE (Relativistic Astrophysics)
- **Panel:** relevance of exascale to mid-range HPC