A Multi-level Scalable Startup for Parallel Applications

Abhishek Gupta, Gengbin Zheng and Laxmikant V. Kalé

Parallel Programming Laboratory
University of Illinois at Urbana-Champaign

ROSS 2011
05/31/11

INTRODUCTION

- High performance parallel machines with hundreds of thousands of processors and petascale performance already in use.
- Parallel startup is challenging - how to start the application on all the computation nodes
- Involves two component
  - *Parallel launching* of appropriate processes on the given set of processors and
  - *Setting up communication channels* to enable the processes to communicate with each other after startup has completed
- Our definition of the startup process is different with the ones that only consider parallel launching
  - e.g. remote execution tools such as GXP and TakTuk

MOTIVATION

- Absence of fast startup mechanisms =>obstacle to the full utilization of high performance computing power by the research community
  - Existing parallel startup mechanism such as those used by Charm++ and OpenMPI take 2 to 4 minutes for startup for 8K processors on Ranger
  - Tremendous SU usage just to startup the application
    - Startup time of 4 minutes for 16K processors → a single experiment on 16K processors results in consumption of more than 1K SUs for application startup.
    - Limits the number of experiments a researcher can perform given the fixed allocation size.

CURRENT APPROACHES

Two types of approaches

1. Assumes the presence of special purpose daemons running on compute nodes to facilitate the startup process
   - e.g. Multi-purpose Daemons (MPD) used for MPICH jobs.
   - Drawback: daemons exist even when no MPI application is running.
2. No special purpose daemons – use a launcher
   - Starts processes on compute nodes using existing daemons such as rsh or ssh and then sets up communication channels among them.
   - As we go up to high core counts, the centralized launcher becomes a bottleneck and imposes scalability limitations

OUR APPROACH

- Multi-level Scalable Startup : Fundamental idea - use of multiple launchers which form a startup tree and reside on different processors.
  - Makes the parallel startup process decentralized
  - Scales well with increasing number of processors.
- Incorporate SMP-awareness to achieve faster startup.
- Introduce the concept of batching of remote shell sessions
- No special daemons (except rsh or ssh daemons) for startup
- Discuss the trade-offs involved in parallel startup using a theoretical model.
- Starting up a Charm++ program on 16,384 cores of Ranger with Ethernet as the underlying communication layer now takes only 25 seconds.
  - SU consumption reduced by an order of magnitude compared with the centralized startup.
  - Outperforms Open MPI startup by a factor of over 8 and MPICH2 startup (using Hydra) by a factor of 4 for 16K cores on Ranger

BASIC METHOD

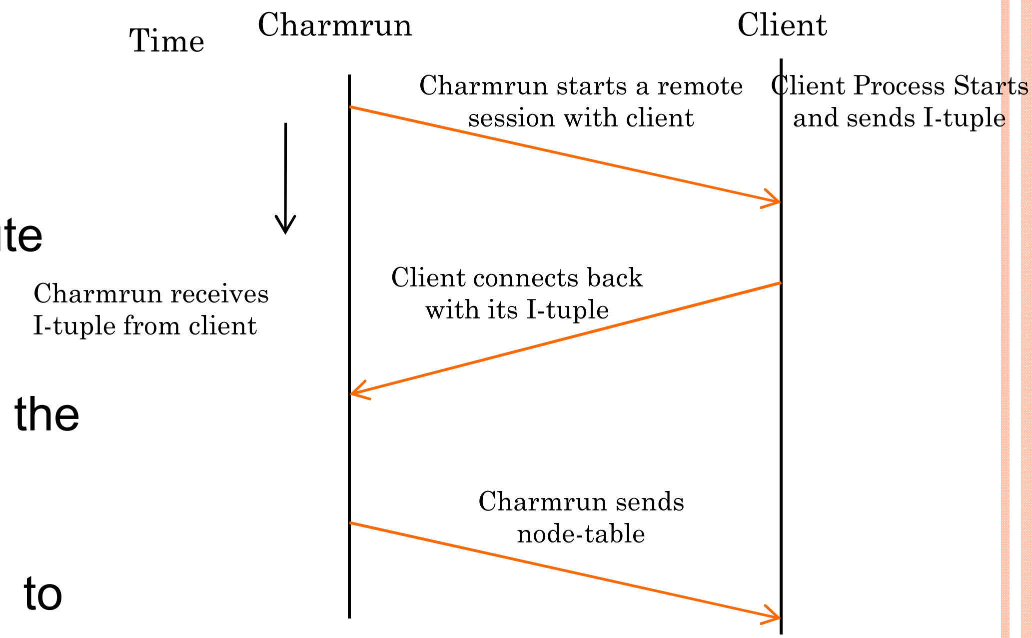
Charmrun
- Launcher
- Process manager

Clients - processes which constitute the parallel application.

nodelist - set of processors where the parallel application will be run

I-tuple – information sent by client to charmrun, used to set up communication channels between the clients.

Node-table – Collection of all I-tuples



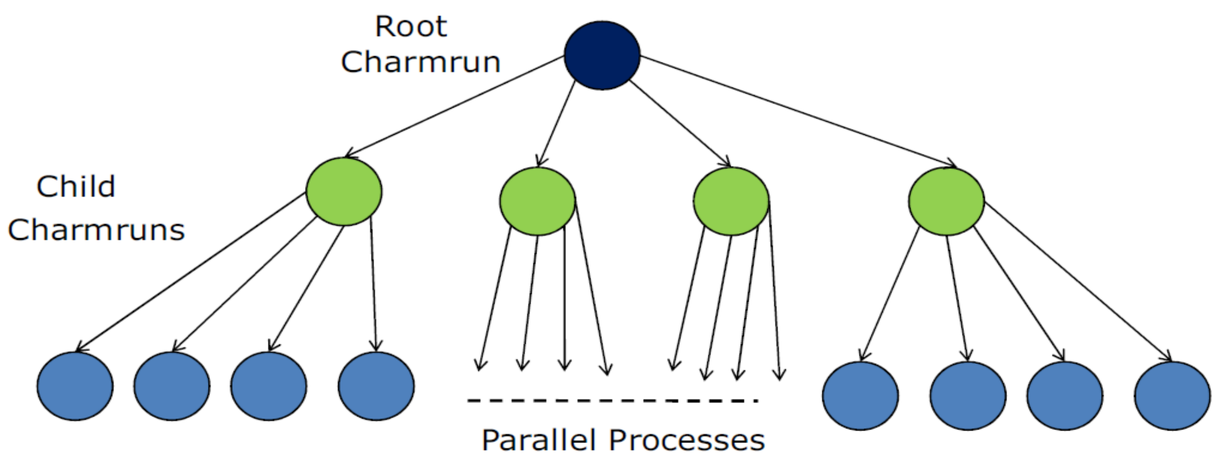
Basic process of parallel startup

SMP-AWARE STARTUP

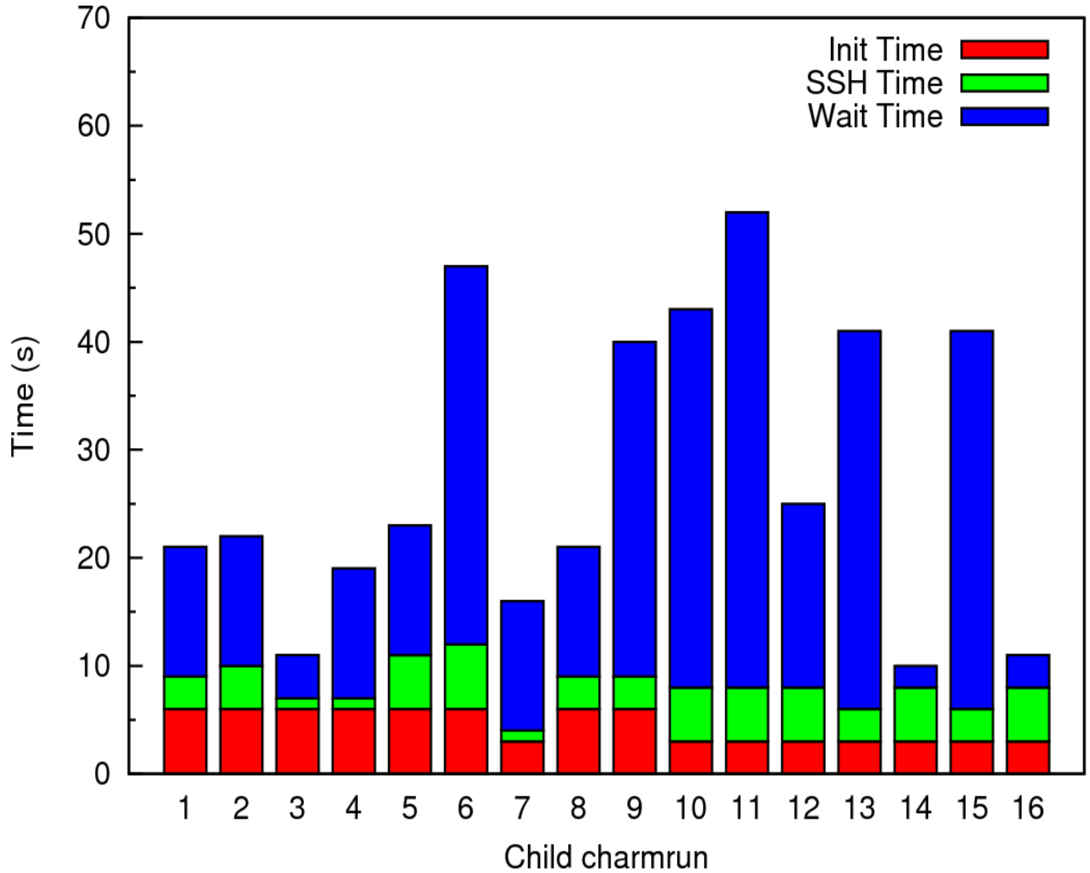
- Linear Startup
  - Charmrun perform a remote shell login to each processor.
- SMP-aware startup
  - Motivation: Most supercomputers and even desktop systems today have multi-core chips
    - Each node has many processor cores. e.g. 8-core, 16- core and 32-core
  - Idea - create only one ssh session per node and spawn all clients from the same ssh session.
  - Also useful when multiple processes need to be launched on a single processor, such as in parallel application testing and debugging.
  - Second phase of startup remains the same as the linear startup.

MULTI-LEVEL STARTUP

- Centralized startup is bottleneck
  1. Charmrun has to start an ssh session with each node.
  2. Charmrun has to receive a message containing an I-tuple from each of the clients
- Multi-level startup



MULTI-LEVEL STARTUP



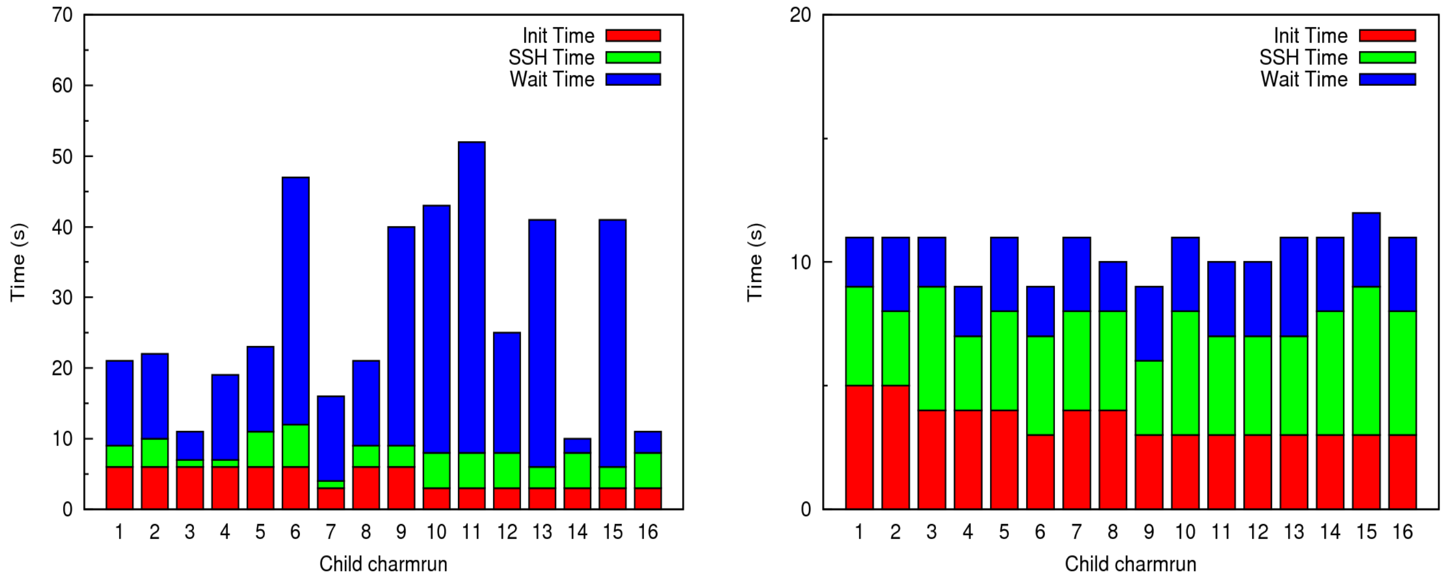
Breakup of time spent in parallel startup using multi-level scheme for 4K processors

MULTI-LEVEL STARTUP - BATCHING

- Variation in startup time small for less number of cores becomes worse with increase in number of cores
- Possibility of network congestion ?

- Batching of remote shell sessions
  - Nodes assigned to a leaf charmrun divided into sets of fixed size.
  - Each leaf charmrun performs ssh to the nodes in current set, waits for the clients to connect back and then performs ssh on the next set.
  - Batch size = number of nodes in one ssh set
- Reduces the total number of messages at any time
  - Better scalability
- Introduces some serialization

MULTI-LEVEL STARTUP - BATCHING



Breakup of time spent in parallel startup using multi-level scheme for 4K processors

ANALYSIS OF STARTUP SCHEMES

Consider a supercomputer with *P* processor cores and *N* nodes.

$$T_{linear} = T_{init} + P \times T_{ssh} + T_{client}$$
$$+T_{send} + T_{nw} + P \times T_{recv}$$

where

T_{init} : Charmrun initialization time (includes getting the list of nodes to start e.g. reading nodelist, starting a server port where clients can send I-tuples etc)

T_{ssh} : Time taken by charmrun to start a rsh or ssh session with a remote node

T_{client} : Time taken by the remote shell to create a new process at the remote processor and load the program executable

T_{send} : Processor sending overhead at a client,

T_{nw} : Network latency for a message,

T_{recv} : Message receiving overhead incurred by charmrun.

$$T_c = T_{init} + T_{client} + T_{send} + T_{nw}$$

ANALYSIS OF STARTUP SCHEMES

$$T_{linear} = T_c + P \times (T_{ssh} + T_{recv})$$

$$T_{SMP} = T_c + N \times (T_{ssh} + c \times T_{recv}) \qquad c = P/N$$

$$\begin{aligned} T_{d-level} = {} & d \times (T_c + k \times (T_{ssh} + T_{recv})) \\ & + k \times (c-1) \times T_{recv} \end{aligned} \qquad d = \log_k(N)$$

k = branching factor and d = depth of the startup tree.

$$\begin{aligned} T_{\text{batched d-level}} = {} & (d-1) \times (T_c + k \times (T_{ssh} + T_{recv})) \\ & + (k/b) \times T_c + k \times (T_{ssh} + c \times T_{recv}) \end{aligned}$$

b = batch size

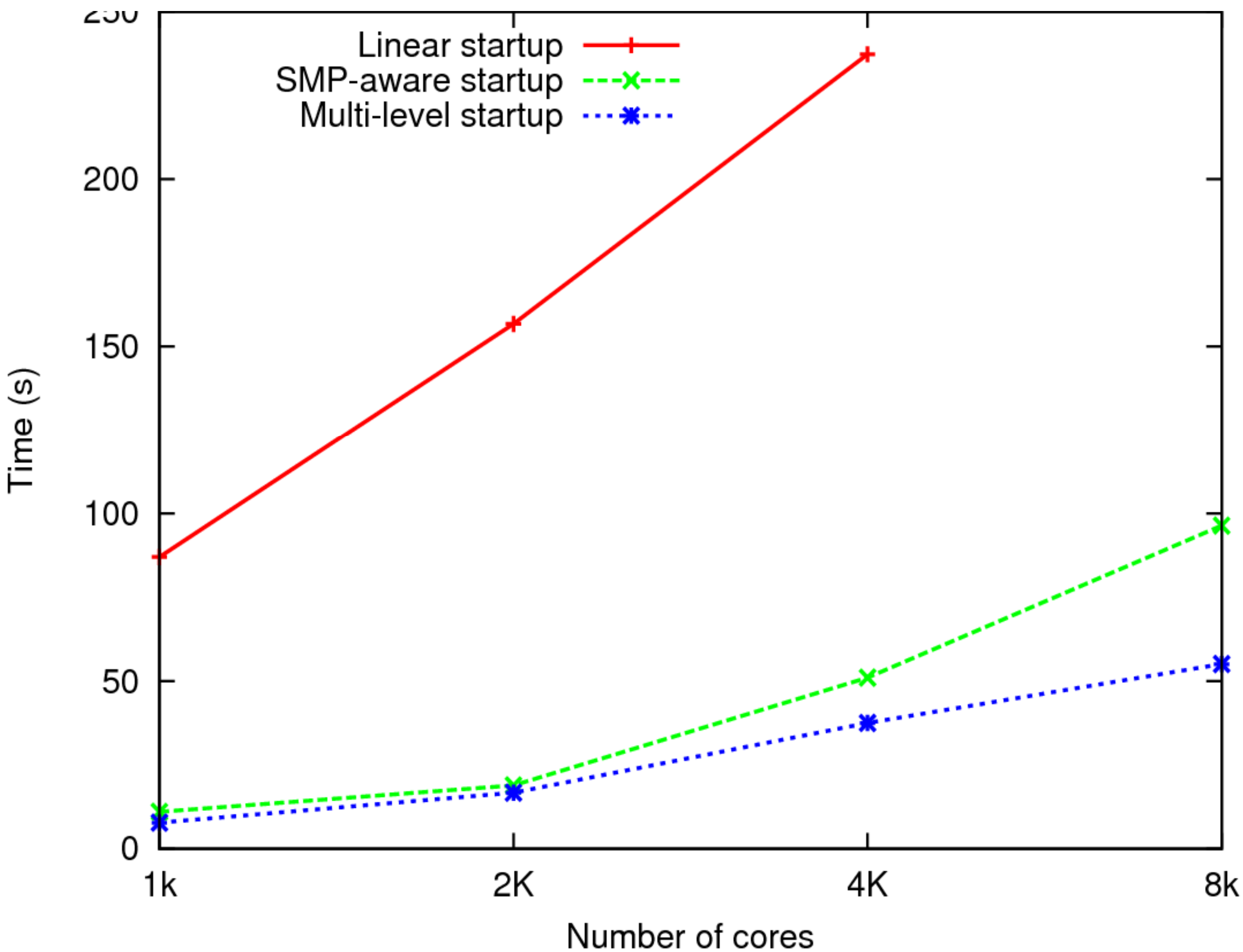
RUNTIME CAPABILITIES

- Process Health and Recovery from Failures
  - Monitor Process health
  - Facilitate fault tolerance – restart protocol
- Support for Scalable Interaction with Parallel Application
  - Parallel debugging – CharmDebug
  - Online performance analysis
  - Simulation visualization - LiveViz

PERFORMANCE RESULTS: PLATFORM

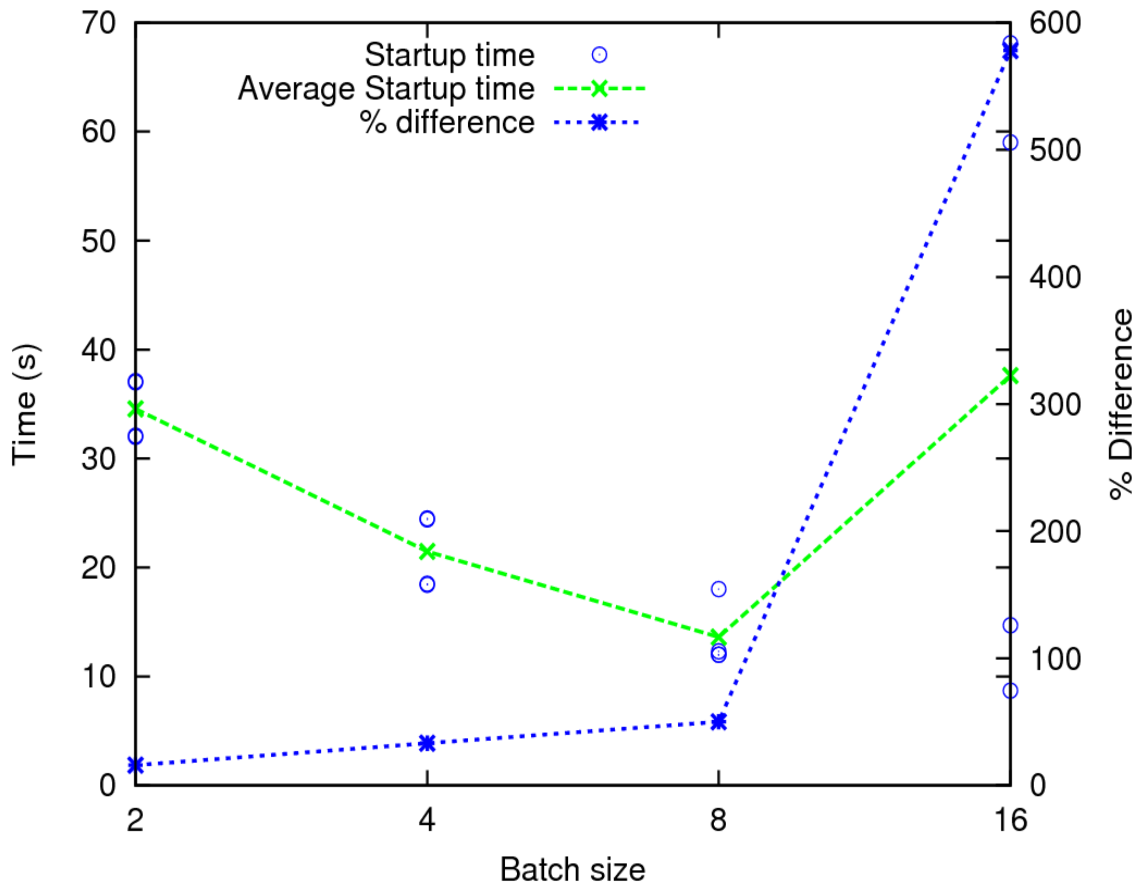
- TACC's Ranger Cluster
  - 3,936 16-way SMP compute nodes providing 15,744 AMD Opteron processors for a total of 62,976 compute cores
- For core counts > 4K, executable cached in each node's memory
- Ethernet as the underlying communication network

PERFORMANCE OF DIFFERENT SCHEMES



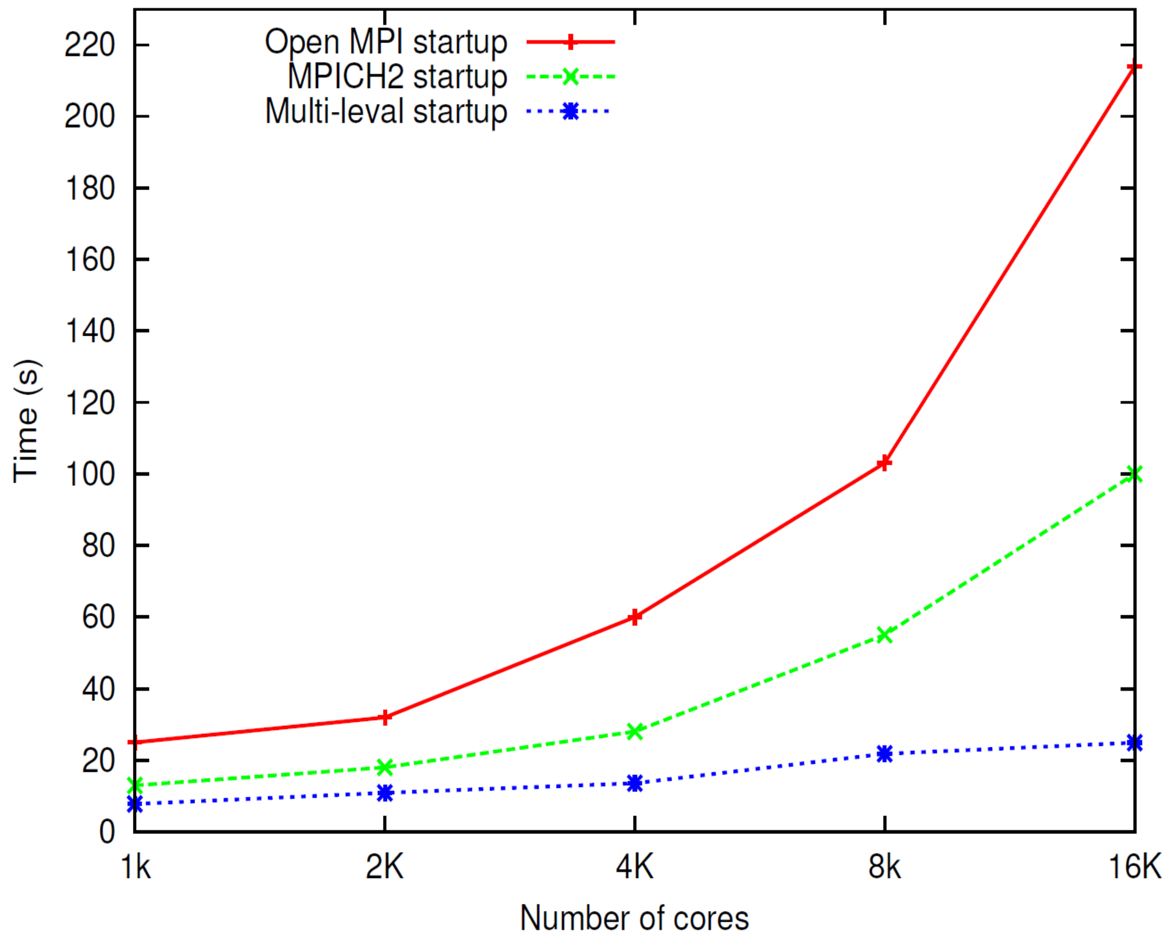
Startup time: Comparison among 3 startup schemes

EFFECT OF BATCHING



Variation in startup time with batch size for 4K processors

Comparison with Open MPI and MPICH2 (Hydra) startup



Startup time on Ranger: Open MPI vs. MPICH2 (Hydra) vs. Multi-level Startup

RELATED WORK

- MPD: Multi-purpose Daemon
  - Special purpose persistent daemons
    - Typically one instance per host in a TCP-connected network
    - Daemons are connected in a ring
  - To run an MPI program, mpirun first connects to the daemon ring in order to start the parallel program and then switches to manager ring in order to control the program
- SLURM
- ALPS
- STORM

RELATED WORK

- Hydra
  - Default process management framework for starting MPI processes for MPICH2-1.3 onwards
  - Uses existing daemons such as *ssh, rsh, pbs, slurm and sge* to start MPI processes
- ScELA
  - Targets multi-core clusters
  - Node Level Agent (NLA) for every node
    - An NLA is used to launch all processes on a node.
    - Similar to SMP-Aware startup
    - Since there is an NLA per node, there is an extra process per node consuming processor cycles

RELATED WORK

- Cplant
- Concurrent launching strategies
  - Fast and scalable distributed machine administration and parallel application development
  - Processes do not need to communicate with each other, hence the second phase of parallel startup - setting up communication channels is not needed
  - TakTuk , GXP

CONCLUSIONS

- Scalable multi-level approach for startup of parallel applications
- Techniques to speed up
  - Parallel launching of appropriate processes on the given set of processors and
  - Setting up communication channels
- Concept of batching of remote shell sessions
- SMP-awareness to further improve scalability.
- Analysis using a theoretical model
- Outperformed Open MPI startup by a factor of over 8 and MPICH2 startup by a factor of 4 for 16K
- Complete solution to the startup of a parallel application and its management during execution.

FUTURE WORK

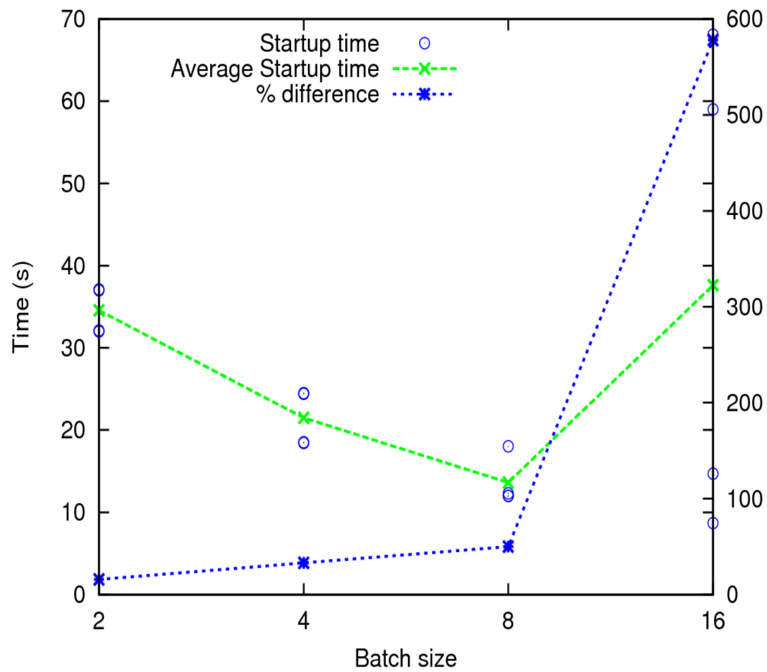
- Startup of parallel application using underlying high-performance interconnects such as Infiniband.
- Lazy establishment of communication channels in the second phase of startup
  - On-demand connection establishment during execution.

ACKNOWLEDGMENTS

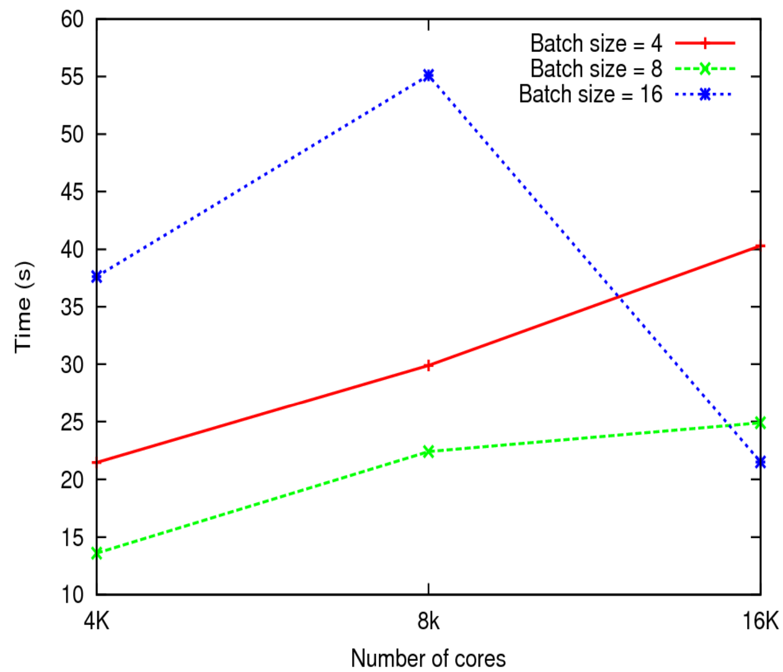
This work was supported in part by NSF grant OCI-0725070 for Blue Waters deployment, by the Institute for Advanced Computing Applications and Technologies (IACAT) at the University of Illinois at Urbana-Champaign, and by Department of Energy grant DE-SC0001845. We used machine resources on the Ranger cluster (TACC), under TeraGrid allocation grant TG-ASC050039N supported by NSF.

Questions

EFFECT OF BATCHING



Variation in startup time with batch size for 4K processors



Startup time vs. number of cores for different batch sizes