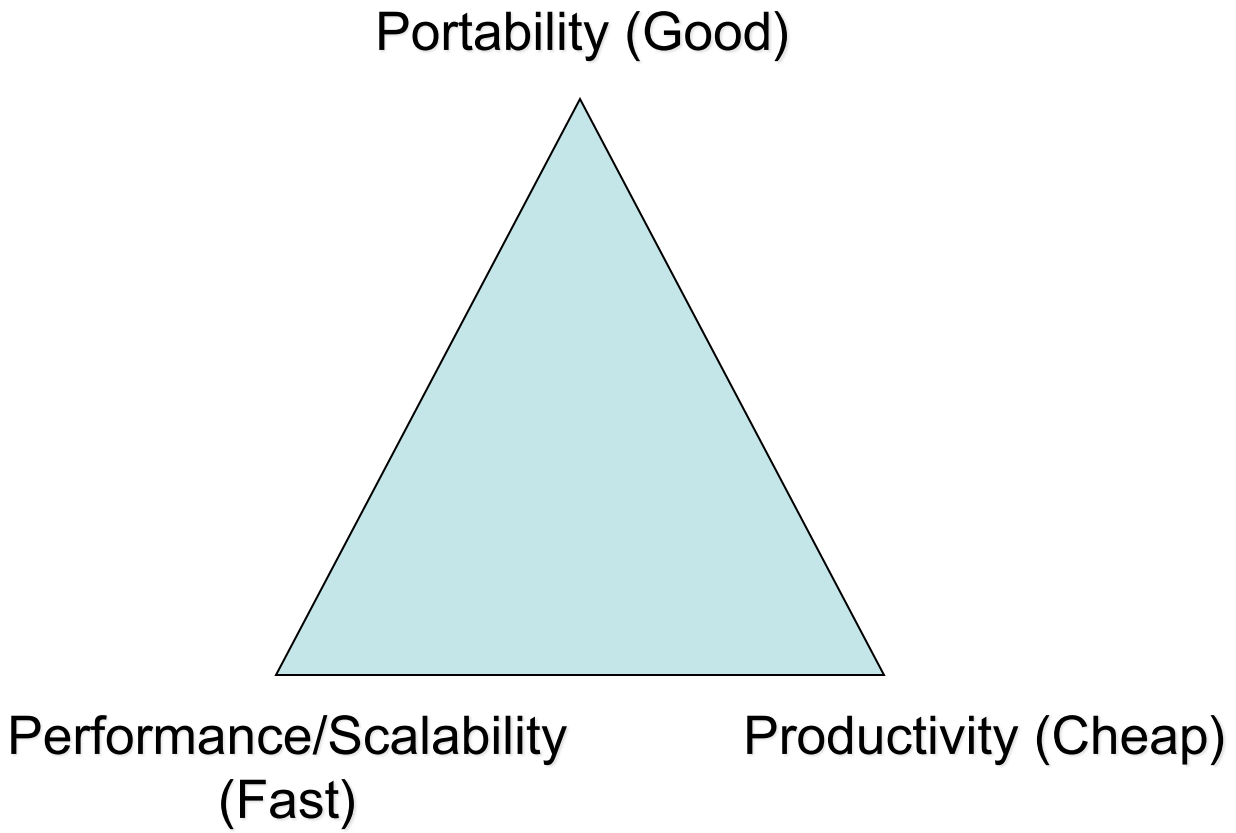
Persistence Working Group
“More Performance, Predictability, Predictability”

June 16, 2010

Tony Skjellum, UAB

MPI/Parallel Program Optimization - PPP Trade Space



http://en.wikipedia.org/wiki/Project_triangle

Wexelblat's Scheduling Algorithm: Choose two: Good; Fast; Cheap.

Applicability/Audience

- Most Existing Users: Every MPI program with regular data parallel data transfers could refactor easily to gain performance
- Current non-adopters: Many users in key areas refuse to use MPI because it clearly adds much too much overhead compared to lower-level transports… MPI adoption can be widened
- Mandated adopters: Certain defense programs are now requiring MPI and delivering higher performance and predictability valuable in embedded scalable systems

Why a Persistence Effort?

- Goals
  - More Performance
  - More Scalability
  - Higher Predictability
- Since MPI-1 we have known there can be faster message passing within the MPI framework… for instance in data parallel, regular or repetitive communication patterns
- Impacts of leaving temporal locality behind grow with path from Terascale -> Petscale -> Exascale
- Can be a concrete and highly useful outcome of MPI-3

Why a Persistence Effort?

- What are identified sources of performance?
  - Temporal locality
  - Simplified critical paths
  - Poly-algorithmic selection (high one time costs)
  - Better mapping of data types to marshalling/ unmarshalling at lowest levels
  - Opportunity to map certain transfers to faster hardware mechanisms
- Planned Transfer used in other systems, they can be much faster than MPI over same network medium on same platform

Why a Persistence Effort?

- Why accept less performance than available if available on an opt-in basis? MPI is about performance after all
- Low performance for programs with temporal locality = prevents adoption in really demanding applications
- Difference in performance compared to low layers lessened = pay less performance penalty at constant portability
- No existing MPI programs are broken
- Improves/Refreshes fundamental parts of standard after 17 years. Can apply to some MPI-2 API too.

Why a Persistence Effort?

- Better than and easier than profile guided optimization
- Expressivity actually makes optimization easier for MPI implementers
- Not layerable on top of MPI, fundamentally a “cut through,” resource lock-down, and reuse paradigm within an MPI stack
- “Orthogonal concept” to rest of MPI-3 development… apply to performance-critical operations
- Fully backward compatible and opt-in basis for use
- Early Implementations / Reluctant Implementers can make “no slower” with only minimal work

Planned Transfer - PT2PT

- PT2PT Partially Supports It (Implementations may Fail to Optimize)
- PT2PT Rule: Any send “kind” with any matching receive “kind” [Revisit “Ready”]
- What effort can be saved?
- Argument “freezing” with MPI_Send_Init() and MPI_Recv_Init()
- Existing standard “all early” or “all late”
- “Rebindable” arguments also desirable for occasional changes to buffers or counts
- “Autoincrementing” addresses also useful
- Eliminates cost of wildcards and tags when not used

PT2PT

- What was proposed in MPI-2
- An API to allow persistent I/O to form a new out-of-band optimized transfer
  - MPI_Bind(send_request, comm)
  - MPI_Bind(req_request, comm)
- Pairwise ordering now (sender, receiver, communicator)
- Groups of binding possible too.
- Creates “channels” with single outstanding message semantics

Game of Life Example

- 2D Decomposition
- 9-point stencil (like 2D finite difference)
- At a process
  - 4 fine-grain communications (single value)
  - 4 medium grain communications (overlap possible)… 2 stride 1, 2 stride (local length)
- Temporal locality:
  - Same pattern repeats each generation
  - Memory alternates (red/black, potentially)
- Possible with existing Persistent Send/Recv
- Use 2 sets to capture alternative memory

Game of Life Example

- You can use persistent now, others use MPI_SendRecv
- With persistent API, impact three things:
  - Plan the short transfers with minimum critical path
  - Plan the longer transfers with overlapped I/O
  - Optimize the non-unit stride transfers (2 of these)
- Special hardware opportunities:
  - Map short transfers to special network
- Other interesting option:
  - Mark entire set of 8 requests as a “bundle”, and indicate bundle will always be scheduled as a single “meta request”

Persistent Collective

- Two kinds: non-blocking, blocking
- Runtime decisions on what algorithm to be used can be done robustly - Can hint about “use vs. reuse”
- Expensive resources (like a new thread) can be done only if needed when needed
- Simplest approach: “freeze arguments early only” as with existing Send/Recv_Init
- More interesting: “ability to rebind” periodically

Persistent Collective

- Path to making “alltoall” operations as scalable as they can be
- Removes need for heroic efforts to infer reuse of collectives by programs and decide to pick expensive algorithms

Other potential big wins for Persistent Collective

- Sparse alltoall, using alltoallv or w… non-participants; one time algorithmic pattern determined at “admission,” less communication each reused time, or all participate for faster or more predictable outcome
- Short allreduces … lock down fastest resources, network, memory, schedule
- All allreduces … pick smarter algorithm knowing that arguments are frozen
- 3D data exchanges … manage large stride datatype “crunching” and data marshalling
- Marking whole groups of transfers as a single schedulable entity (“MPI_Start_Init” persistence).. More general protocols per transfer possible than MPI-3 sparse collectives … friendlier to layered libraries too

Persistent Collective Example

- Classic Example:
  - Overlap dot product computation with other linear algebra in conjugate gradient code
  - Example exists all the way back to MPI-1 and IBM extensions to MPI
  - Non-blocking faster (also a non-blocking win)
- Short messages reductions could be mapped to special network if made persistent - their own independent order from other collectives, optimize lock down of limited resources

Persistent Generalized Requests

- Generalized Requests Good for Certain Applications (like layered libraries)
- Being able to reuse them is better
- If you have to create them each time, limits their usability
- Persistent Generalized Requests Open New Opportunities for Standard Extension by applications and middleware for their own needs
- Some MPI-2 Standard Notes (See Section 8.2)
  - “The goal of this MPI-2 extension is to allow users to define new nonblocking operations.”
  - “It is tempting to also define an MPI standard mechanism for achieving concurrent execution of user-defined nonblocking operations. “

Applicable to Parallel I/O

- Transfers with autoincrementing pointers
- Transfers where significant data reorganizations have to happen on one side or the other of an I/O request
- Transfers with complex data types
- Situations where preallocation of buffers (e.g., special memory) is helpful

Applicability to 1-sided/RDMA

- Temporal locality in regular programs/ transfer patterns
- To the extent 2-sided have temporal locality, regular one-sided writes/reads can benefit too
- Long transfers, potentially with autoincrementing and round-robin buffer pointers (very useful use case)
- Crush(crunch) complex datatypes; map to DMA hardware
- Plan for reuse for bulk transfers
- Allocate and lock down limited resources statically for fine-grain operations
- Use of special memory implicitly rather than explicitly?
- Allow push-down of more fine-grain protocol to co-processors?

Previous Efforts - Significant

- Proposed and Prototyped in MPI-2 … like non-blocking collective was left out … fully prototyped and discussed in MPI-2… didn't excite the participants at the time (emphasized MPI-1 portion of standard)
- Defined, refined and implemented multiple times in MPI/RT, lots of experience using such APIs on real systems (e.g., IB, RapidIO, embedded Myrinet)… fast as vendor middleware…
- Discussed in conjunction with non-blocking collective in a 1997 paper at PTPTA

Next Logical Steps

- Propose API for each of three categories
  - Point to Point
  - Generalized Requests
  - Collective
- Show example code
- Explain why important for performance
- Motivate further where savings occur

Summary

- Enables programs that have temporal locality to run faster (e.g., lower latency send/receive mapping to specialized hardware better)
- Supports higher predictability when used
- Enables expensive resource decisions to be deferred till needed in key circumstances
- Supports polyalgorithmic algorithmic decisions at runtime (fixed vs. variable cost)
- Expands audience for MPI
- More performance and predictability after 17 years of MPI-1 and ~13 years of MPI-2.
- “Orthogonal” concept to rest of MPI-3 development.

Some References

- “High Performance MPI Paper”, 1997…
  - http://www.cis.uab.edu/hpcl/publications/documents/Message-Passing/fastmpi.pdf
- MPI/RT Standard:
  - www.mpirt.org
  - Anthony Skjellum, Arkady Kanevsky, Yoginder S. Dandass, Jerrell Watts, Steve Paavola, Dennis Cottel, Greg Henley, L. Shane Hebert, Zhenqian Cui, Anna Rounbehler: The Real-Time Message Passing Interface Standard (MPI/RT-1.1) . Concurrency - Practice and Experience (CONCURRENCY) 16(S1):0-322 (2004)

Post-talk follow-ups

- Tony will get with folks from IBM and elsewhere who have interest in "Planned transfer"…
- Existing proposals from MPI-2 will be revived and shown in October
- No action proposed for Stuttgart