Implementing a Global Address Space Language on the Cray X1

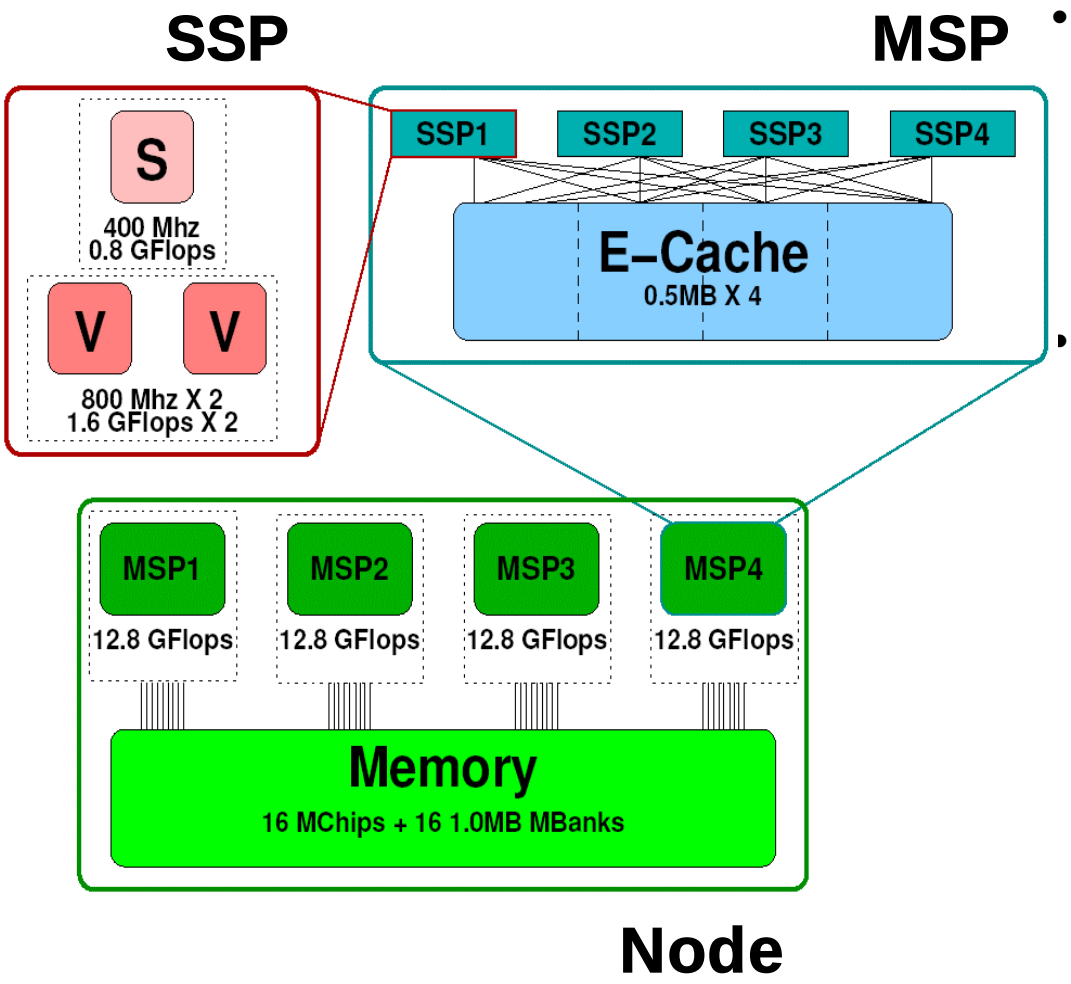
Christian Bell and Wei Chen

Why UPC on the Cray X1 ?

- Supercomputers are mounting a comeback
  - "It's all about sustained and peak performance"
- Parallel Vector systems claim to pack enough features to narrow the gap in sustained and peak performance through vector processing and . . .
  - Compiler-assisted Multistreaming (aggregating vector pipelines)
  - Non-uniform shared accesses
  - Hardware assisted strided scatter/gather accesses
  - Caching local vector accesses for performance
  - *Not* caching remote vector accesses for cache coherence
  - **. . . native support for Global Addressing**

The Cray X1 Architecture

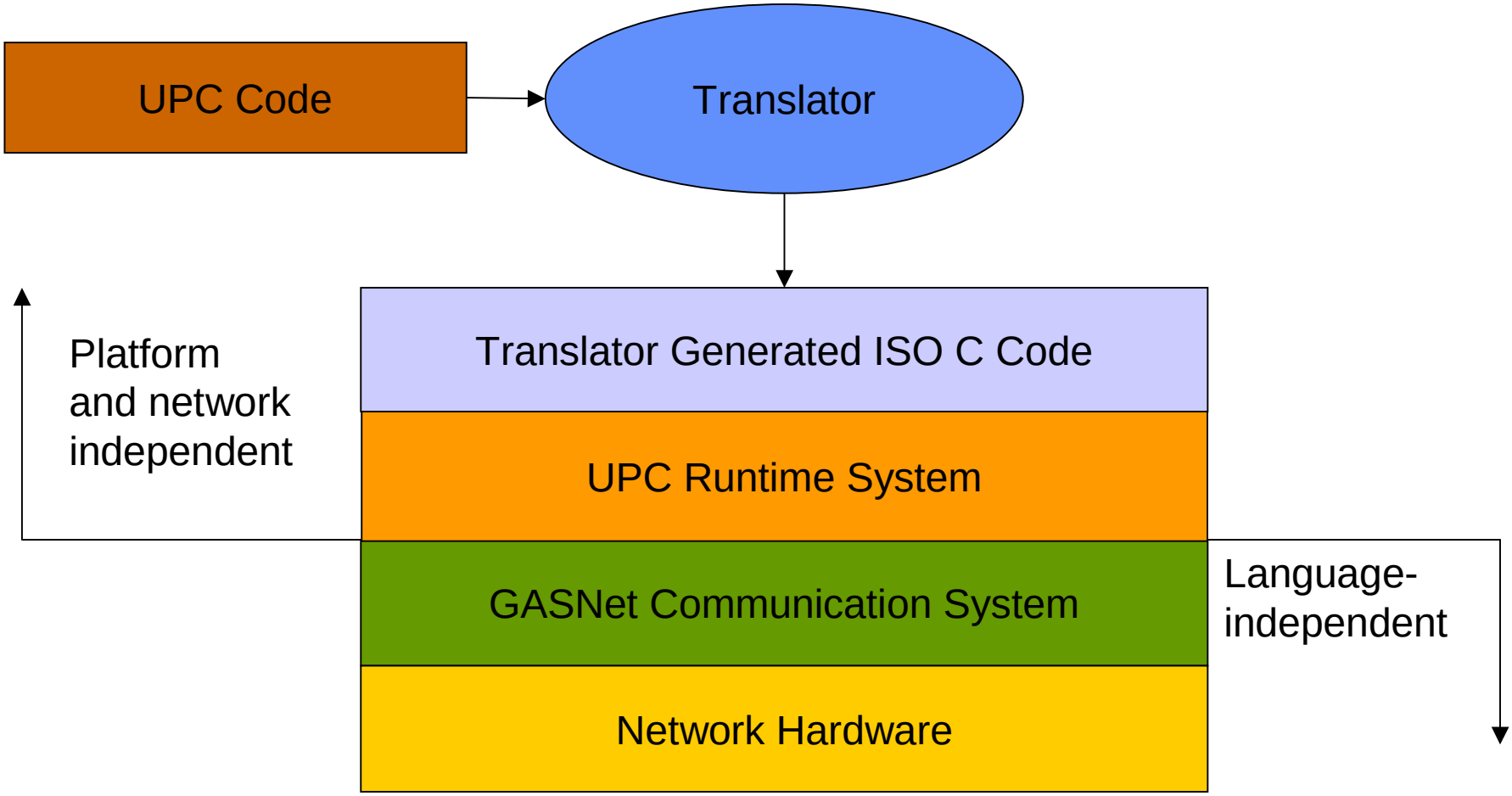
Rich in features, but also in design and programming complexity. . .



- **Two modes of execution**
  1. SSP mode: single-streaming up to 16 SSPs/node
  2. MSP mode: multi-streaming (4 MSPs =16 SSPs/node)
- **Two programming models**
  1. Single Cray X1 node: Shared-memory over uniform memory accesses (pthreads, OpenMP)
  2. Multi-node: Distributed memory between non-uniform memory accesses with no remote caching (MPI, shmem, CoArray, UPC)

Overview of the UPC Compiler

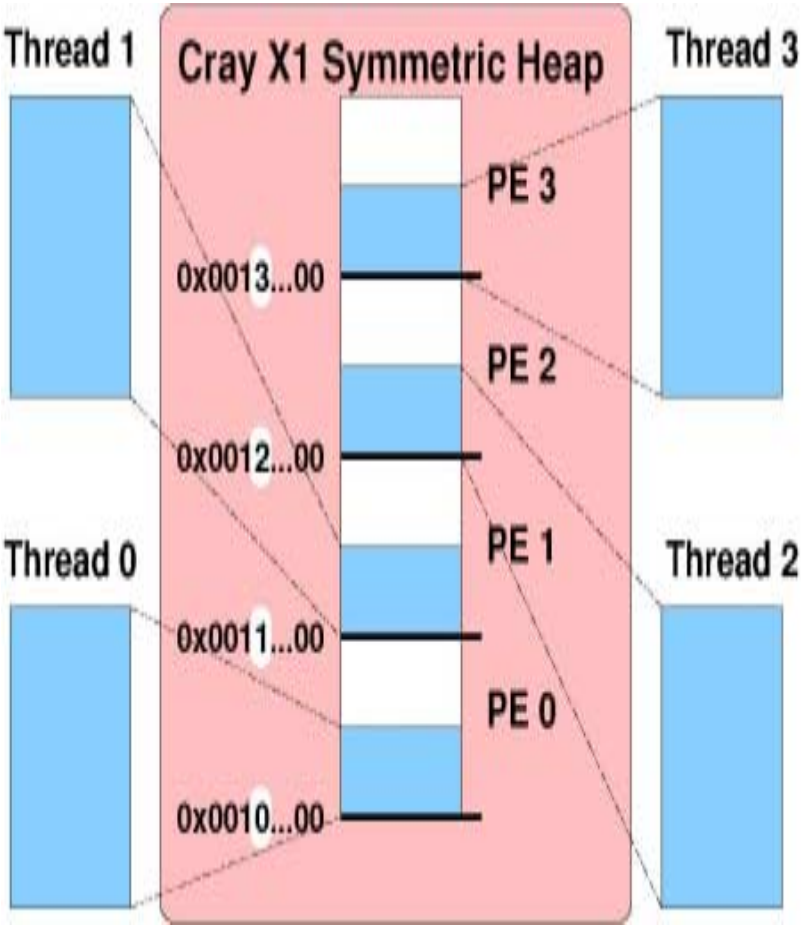
Two Goals: Portability and High-Performance



A Portable GAS Language Implementation on X1

- The X1's network is integrated seamlessly with each X1 node
  - Communication is *implicitly* triggered through a memory centrifuge
  - Network is abstracted from both application and system programmers
  - Our portable compiler (through GASNet) typically targets explicit communication interfaces

- Vector processing makes performance tuning rather difficult,
  - Vectorizing sequential code
  - Vectorizing fine-grained communication



Cray X1 Memory Centrifuge

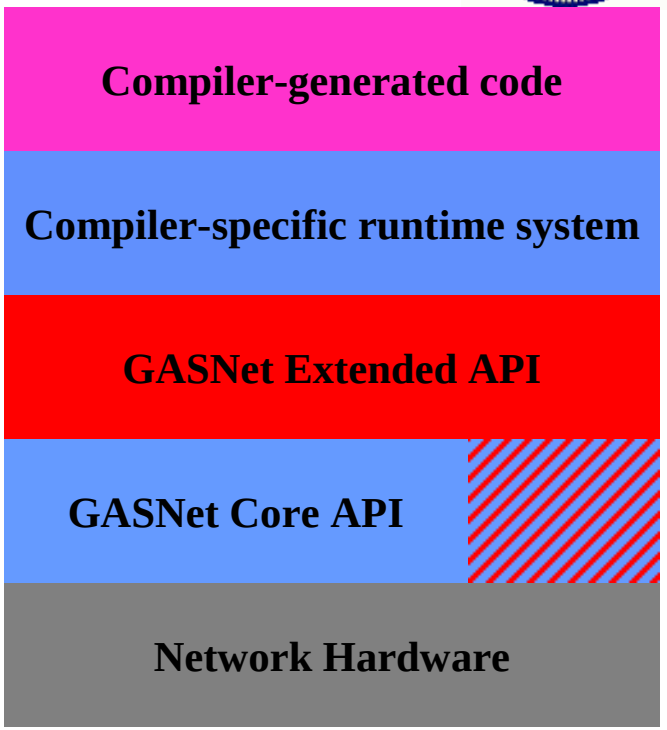
GASNet Communication System-Architecture

GASNet offers expressive put/get primitives

- gets/puts can be blocking or non-blocking (explicit with handles or implicit globally/region-based)
- Transfers can be memory-to-memory or memory-to-register
- Synchronization can poll or block
- Allows expressing complex split-phase communication (compiler optimizations)

2-Level architecture to ease implementation:

- Core API
  - GASNet infrastructure allowed 2-day port
- Extended API
  - Initially target shmem
  - Current revision is tuned especially for the X1 with shared memory as the primary focus (minimal overhead)



GASNet Extended API – Ruling out Cray *shmem*

- Cray Inc.: *shmem* is the “right way” to program the X1 for distributed applications
- Initially targeting Cray *shmem* presented some problems:
  - *shmem* has limited synchronization mechanisms
  - *shmem* gets are entirely blocking
  - *shmem* calls within loops shut down the vectorizer
  - *shmem* prevents integration of global communication in vector computation loops – still bulk synchronous programming style *shmem* pays an address translation cost in every call
- Summary: *shmem* cannot leverage full capability of the hardware for X1 and therefore is *not* a good compilation target for GAS languages
- Alternative: teach global pointer representation to GASNet and/or GASNet clients and bypass *shmem* restrictions altogether

GASNet Extended API – Using Cray X1 global pointers

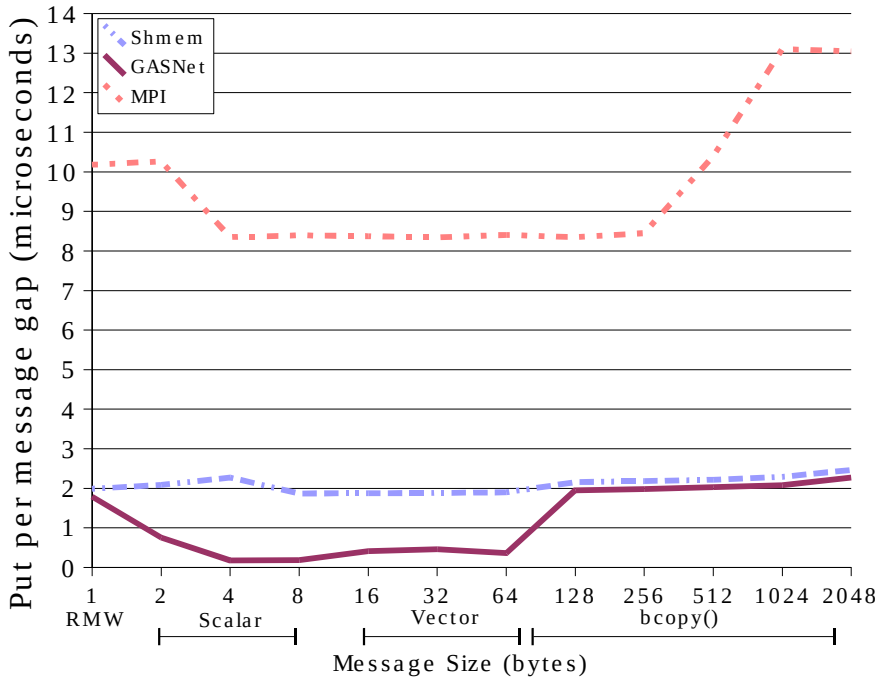
- Alternative: manipulate global pointers directly
  - Push the translation into the client where it can be optimized more efficiently
  - X1 offers no user-level vector *operations*: Cray C schedules vector *assembly instructions* over these global pointers based on translated ISO C
  - GASNet put/get interface is now fully inlinable, hence amenable to Cray vectorization within inner loops
  - Translate get/put into global load/store instructions to allow some overlap at the instruction level
- Next challenge: GASNet is now vector-friendly, the remaining burden lies on the next software layer (UPC runtime system)

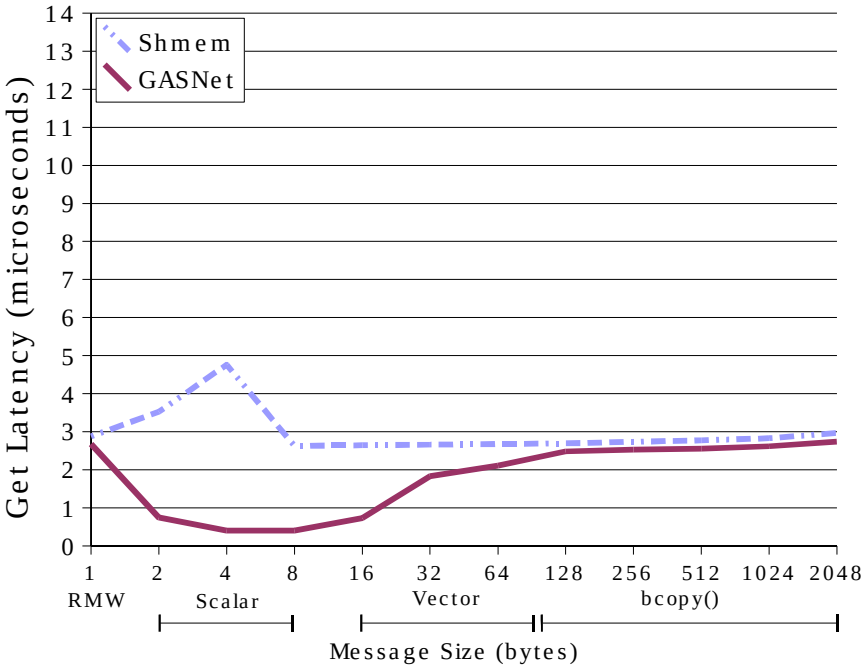
GASNet and X1 memory operations

- Problems with synchronizing memory operations
  - X1 offers a global memory barrier (*gsync*) while GASNet has a rich interface for individually synchronizing operations (semantic mismatch)
  - X1 vectorizer disallows memory barrier within loops
  - No flexible communication scheduling possible if GASNet has no control over individual operations (*. . . giving a sledgehammer to an ear surgeon)*

- Solution: avoid the use of *gsync* for fine-grained communication
  - No sync except for strict memory accesses
  - Encourage clients to use GASNet's implicit non-blocking operations and push the sync out of the loop

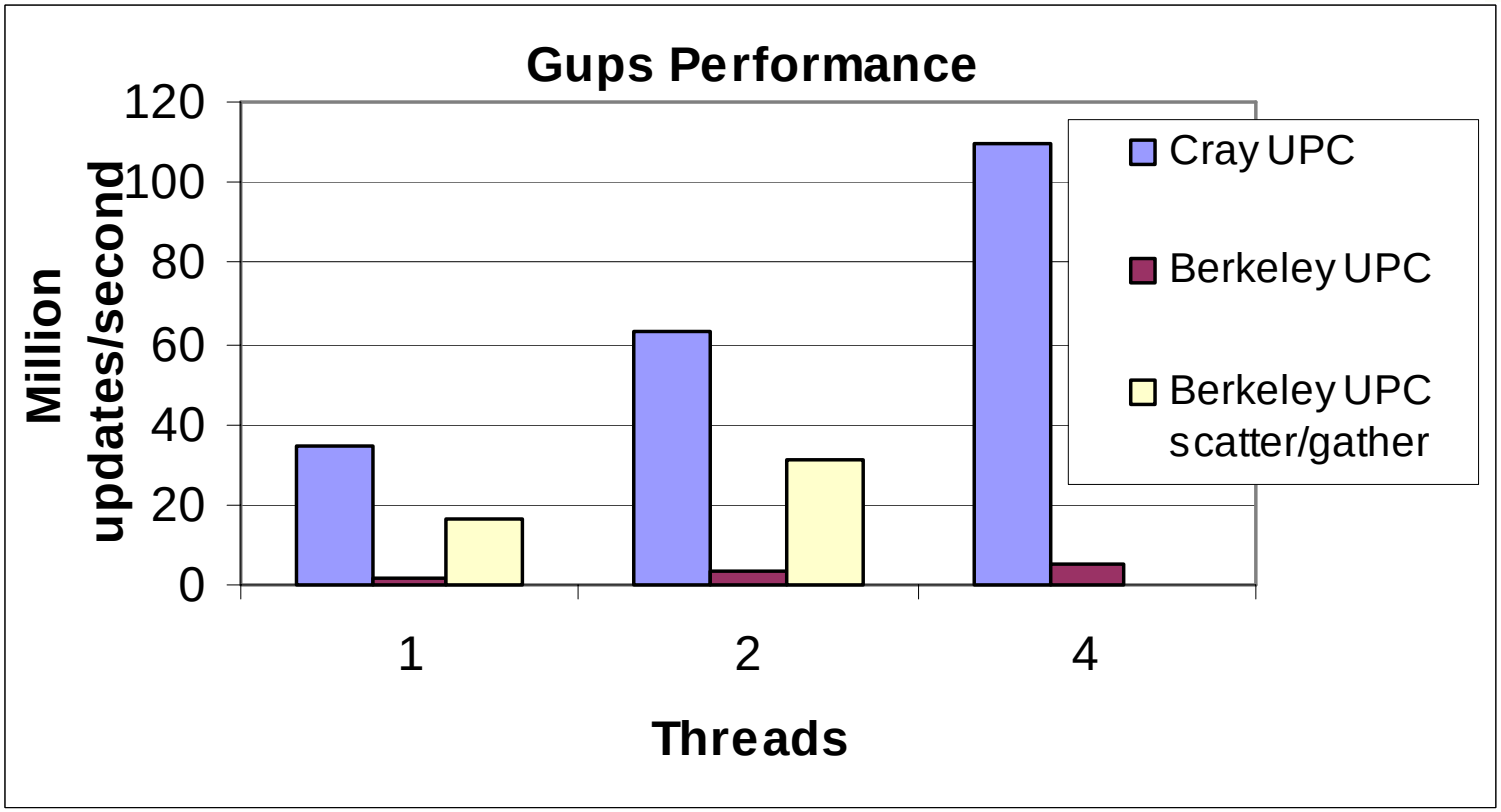
GASNet/X1 Performance





- GASNet/X1 improves small message performance over shmem and MPI (smaller is better!)
- GASNet/X1 communication can be integrated seamlessly into long computation loops and is vectorizable
- GASNet/X1 can operate directly on global pointers (no translation)

Fine-grained Irregular Accesses – UPC GUPS

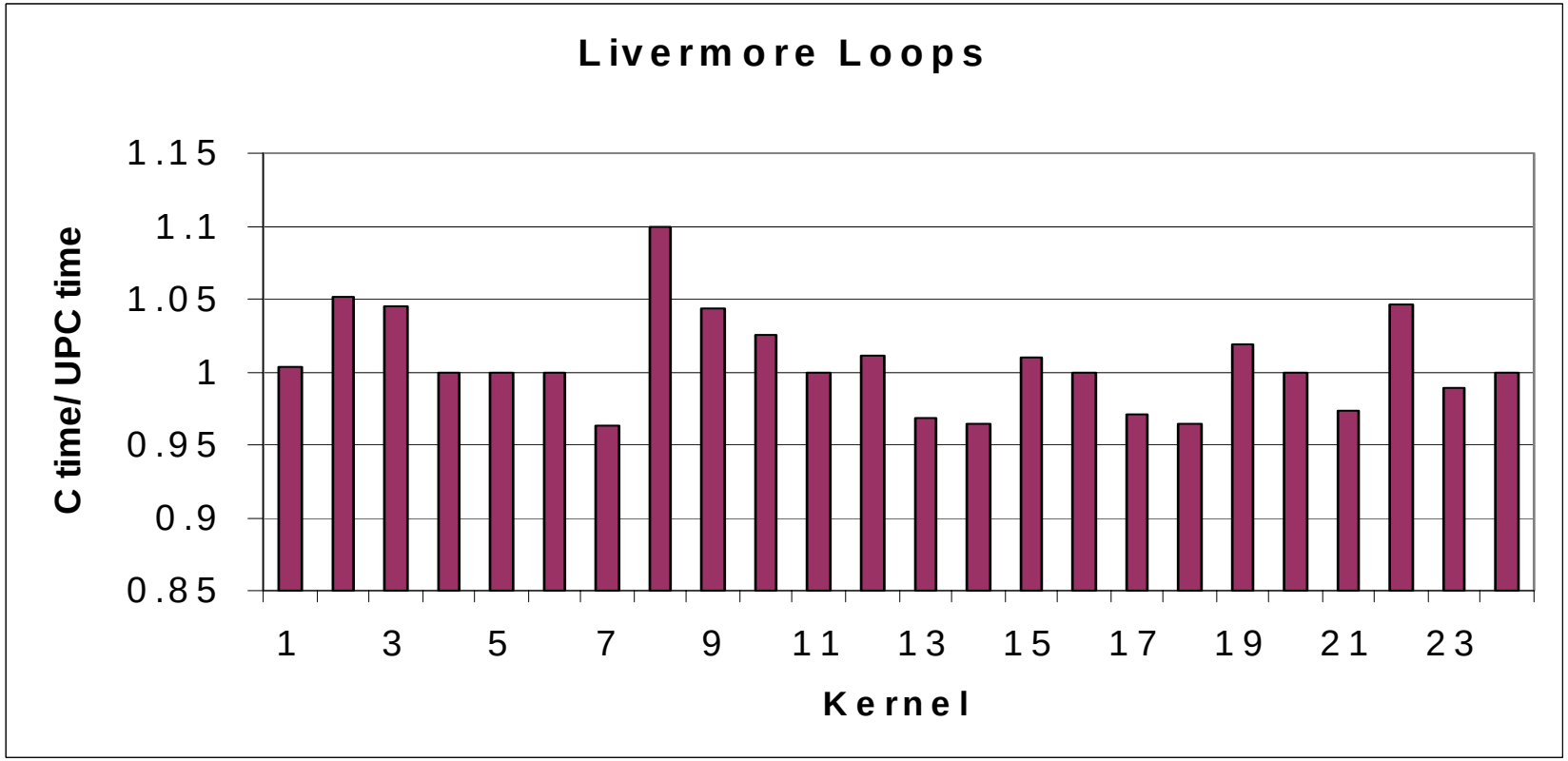


- Hard to control vectorization of fine-grained accesses
  - temporary variables, casts, etc.
- Communication libraries may help

Serial Performance

- It's **all** about vectorization
  - C is a poor compilation target for vectorization
  - Cray C highly sensitive to changes in inner loop
- Problem easier for C/Fortran based GAS languages
  - Just keep code syntactically close to original source
  - Assuming the user has done the application work to vectorize
- Code generation strategy
  - keep IR at a high level (e.g., keep array nodes, field accesses)
  - preserve source level pragmas
  - preserve restrict qualifiers

Evaluating Source-to-Source Translation in UPC

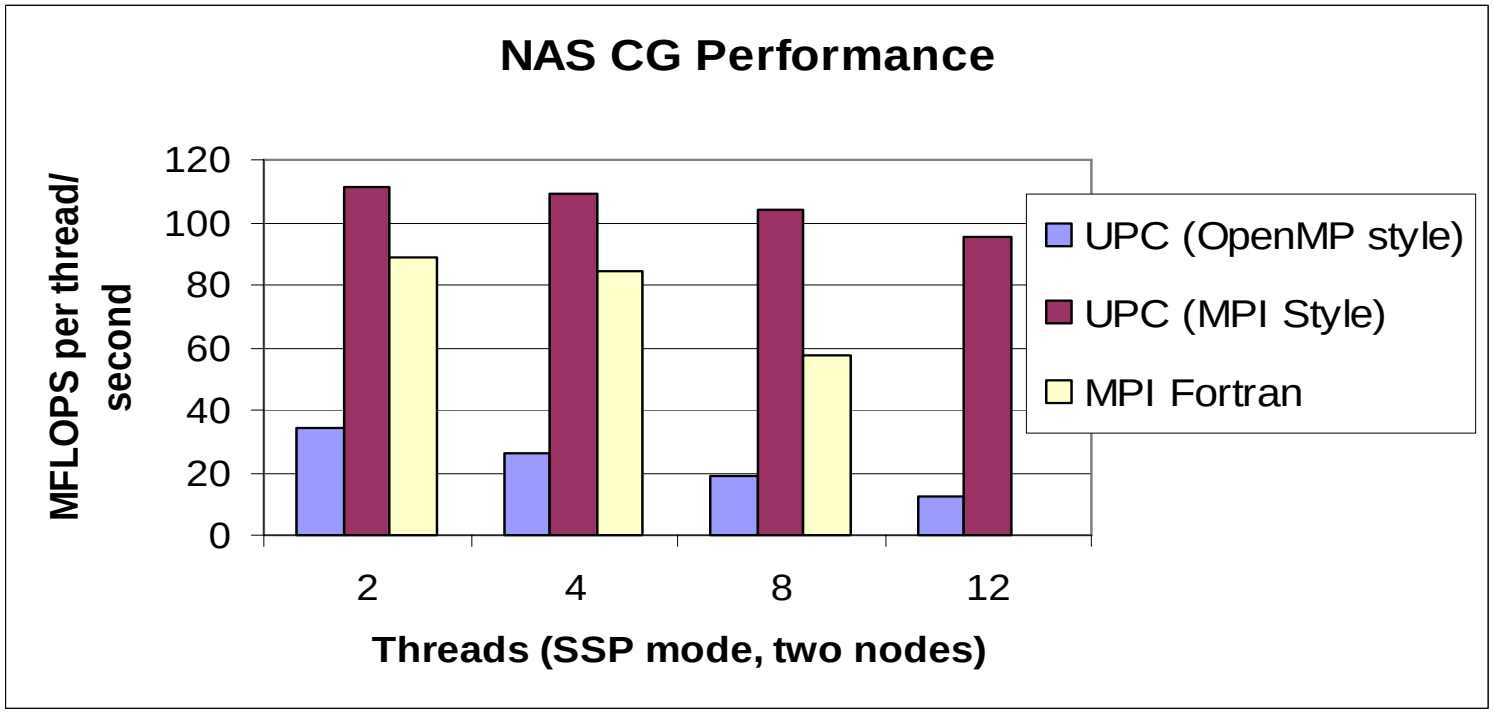


- Translator generated C code can be as efficient as original C code
- Source-to-source translation a good strategy for portable GAS language implementations

Evaluating Communication Optimizations on Cray X1

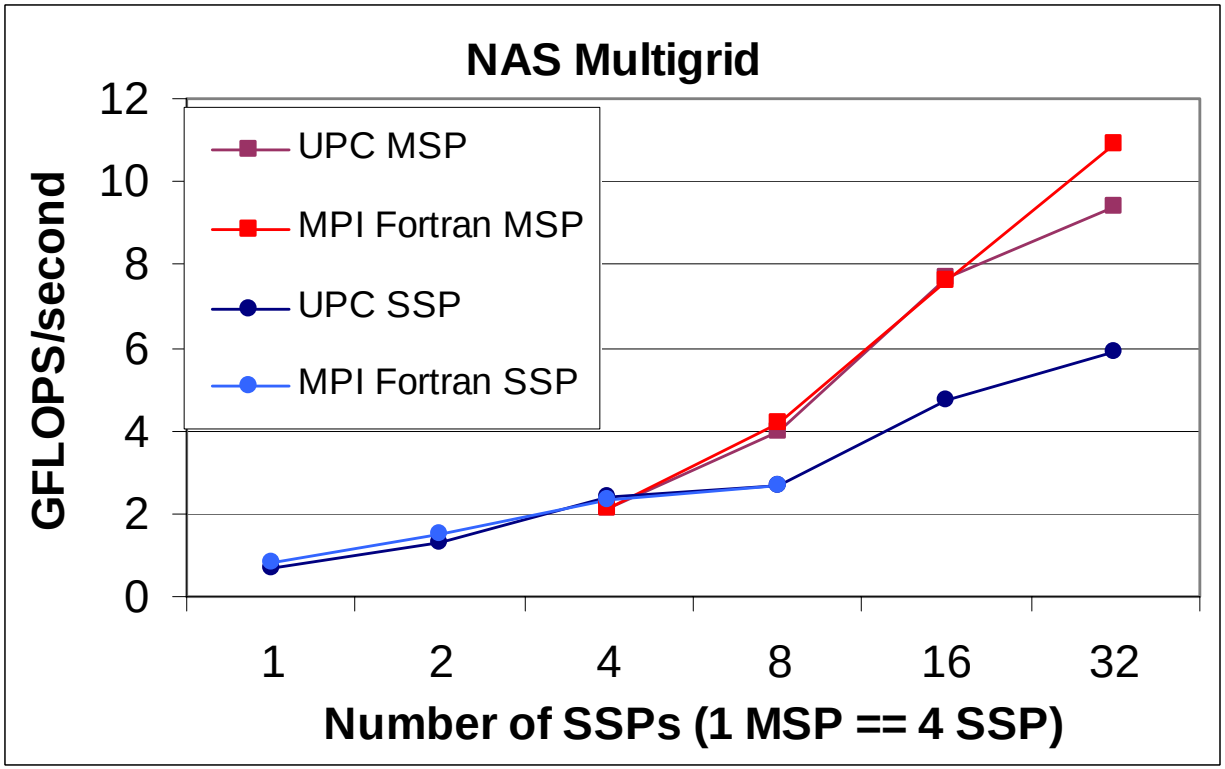
- Message Aggregation
  - LogGP model: fewer messages means less overhead
  - Techniques: message coalescing, bulk prefetching
  - Still true for Cray X1?
    - Remote access latency comparable to local accesses
    - Vectorization should hide most overhead of small messages
    - Remote data not cacheable – may still help to perform software caching
  - Essentially, a question of fine-grained vs. coarse-grained programming model

NAS CG: OpenMP style vs. MPI style



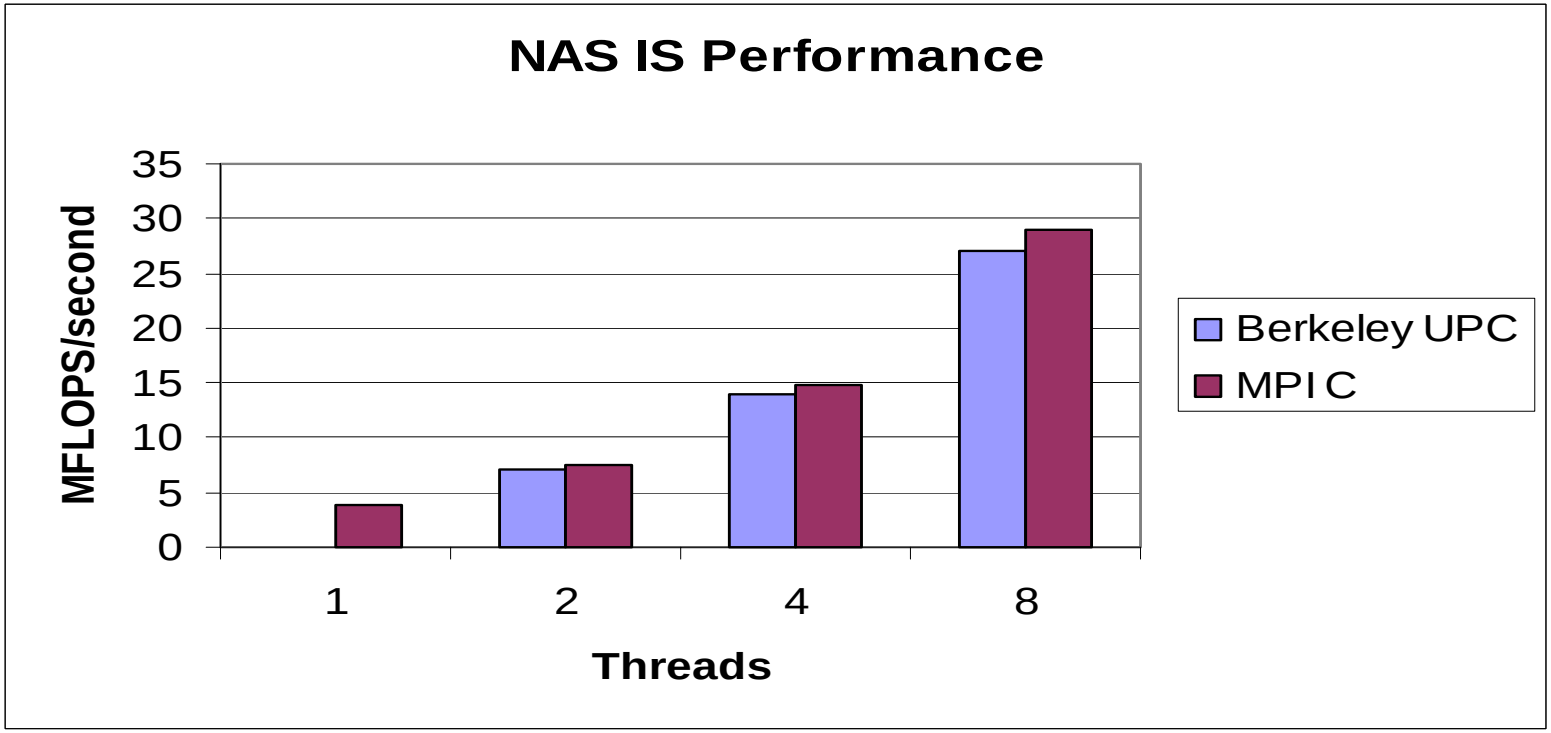
- GAS language outperforms MPI+Fortran (flat is good!)
- Fine-grained (OpenMP style) version still slower
  - shared memory programming style leads to more overhead (redundant boundary computation)
- GAS languages can support both programming styles

Multigrid



- Performance similar to MPI
- Cray C does *not* automatically vectorize/multistream (addition of pragmas)
- 4 SSP slightly better than 1 MSP, 2 MSP much better than 8 SSP
  - cache conflict caused by layout of private data
  - serious design flaw in our opinion

Integer Sort



- Benchmark written in bulk synchronous style
- Performance is similar to MPI
- Code does not vectorize – even the best performer is much slower than cache-based superscalar architecture

Conclusion: We have a GASNet conduit on Cray X1!

+ Provides integrated *application* software
+ Good performance for individual memory operations
+ Transparent communication through global pointers
- Poor user-level support for remote sync operations (no prefetching or per-operation completion mechanisms)
- Heavy reliance on vectorization for performance – great when it happens, awful otherwise
- Sensitive to translated code (slow scalar processor)
- ***Software architecture is not extensible for third-party library or system software programmers***

± Semantic mismatch between GASNet and platform – we're hoping the X2 can address our concerns