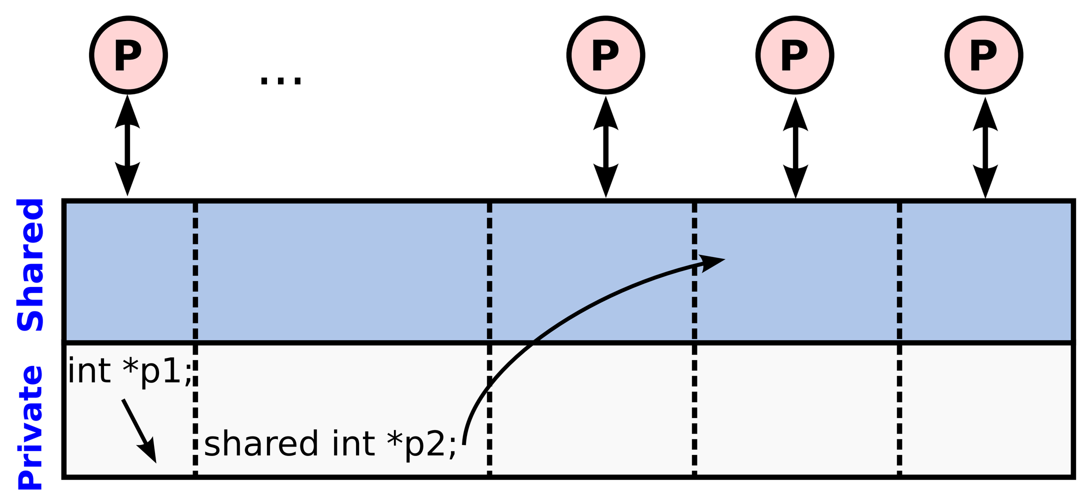
Improving Performance of All-to-All Communication Through Loop Scheduling in PGAS Environments

Michail Alvanos
Gabriel Tanase
Ettore Tiotto
Montserrat Farreras
José Nelson Amaral
Xavier Martorell

Barcelona Supercomputing Center, c/ Jordi Girona 31, 08034 Barcelona, Spain

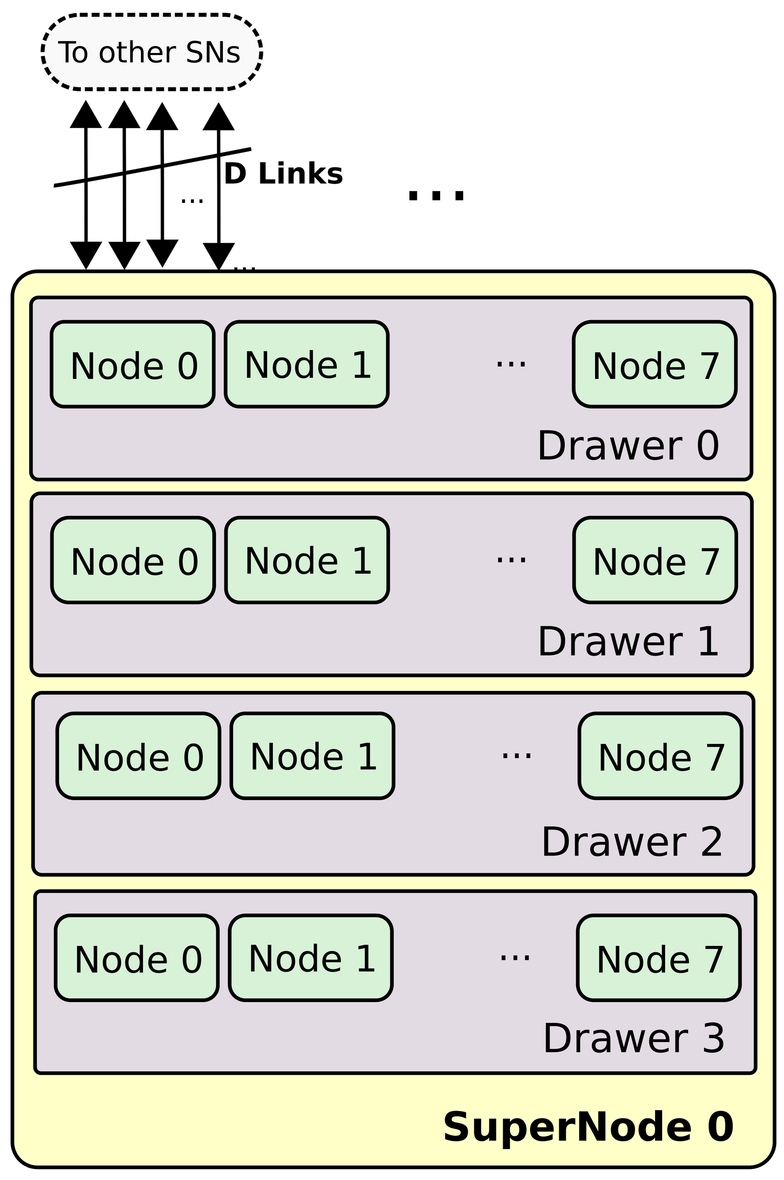
Partitioned Global Address Space (PGAS)

- Goal: Simplicity of shared-memory...
- ...with efficiency of the message-passing paradigm
- Shared or distributed memory
- Unified Parallel C (UPC): ISO C 99 extension



Motivation

- All-to-all communication suffer from node oversubscription
- Manual or compiler code optimization is required



We propose

- Loop scheduling for better network utilization

Platform

- XL UPC framework
- Power775: 32 Nodes x 32 Cores
- Hub-Chip: High-Radix topology

Loop Scheduling

- Core idea: schedule the accesses to ensure that each thread will not access the same shared data
- Spread communication across all the nodes
  - Avoid node over subscription
  - Avoid network hotspots

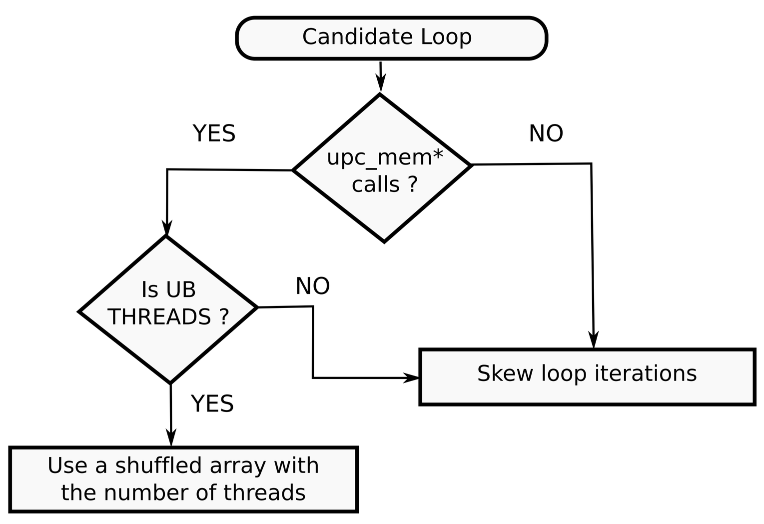
Loop Transformation Approaches

- Skew loops and start from a different point
  - NEW_IV = (IV +MYTHREAD×Block) % LOOP_UB;

  Where $Block = \frac{SIZE\_OF\_ARRAY}{THREADS}$
- Strided accesses: starting from a different Node
  - NEW_IV = Block×(IV ×33+MYTHREAD) % LOOP_UB;
- Random shuffled: when the upper bound is the number of threads

Compiler loop transformation

- The compiler categorizes the loops in two categories based on the loop upper bound and type of accesses
- The compielr inserts the new code and replaces any occurance of the induction variable



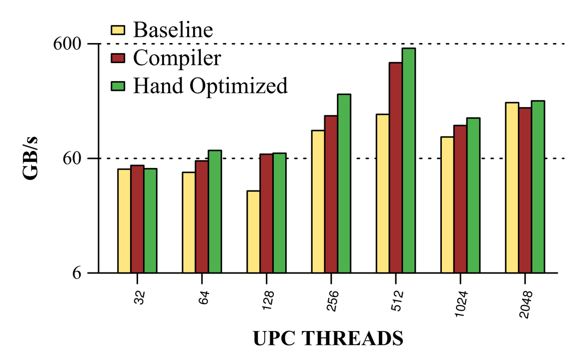
Example Transformed Code:

```
shared [N/THREADS] double X[N];
shared [N/THREADS] double Y[N];

// Example All-to-all
void memget_threads_rand(){
   uint64_t i=0, block = N/THREADS;
   double *lptr = (double *) &X[block*MYTHREAD];
   uint64_t *tshuffle = __random_thread_array();

   for (i=0;i<THREADS;i++){
      uint64_t idx = tshuffle[i];
      upc_memget( lptr, &Y[idx*block],
                  block*sizeof(double));
   }
}
```

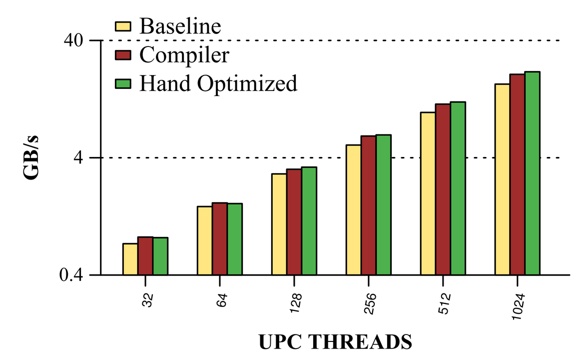
Current state & ongoing research

- Demonstrated performance improvements using several applications, further tuning underway
- Microbenchmarks: slightly lower performance that the manual optimized benchmark
- NAS FT achieves a speedup between 3% up to 15% due to all-to-all transpose
- Bucket-sort achieves 3-8% performance grain except when running with 32 UPC threads
- The optimization is effective and scales well, when the communication takes a considerate amount of time
- Current research aims to 1) decrease the overhead of compiler 2) Find more optimal traffic scheduling 3) Cover more cases

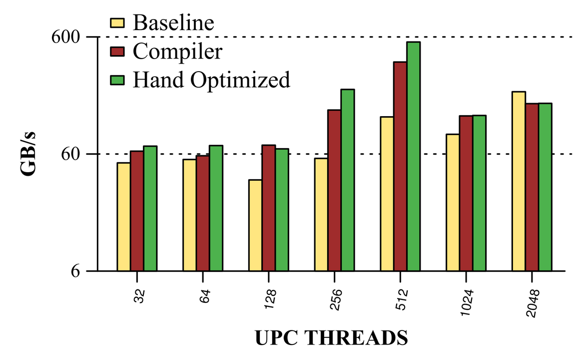
UPC Memput



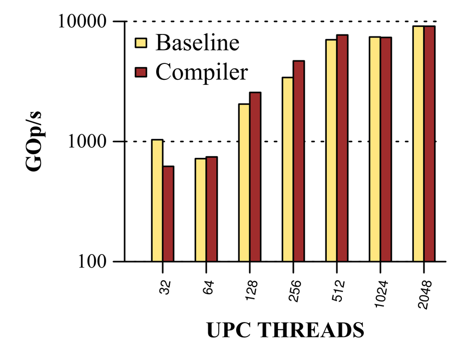
Fine grained get



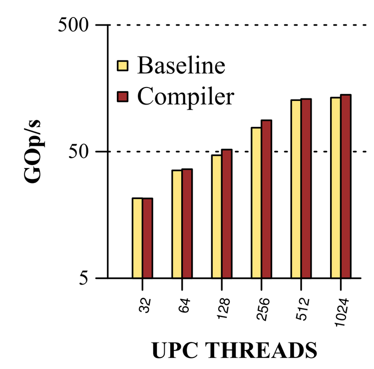
UPC Memget



Bucketsort



FT



Barcelona Supercomputing Center — Centro Nacional de Supercomputación
UPC — Departament d'Arquitectura de Computadors — UNIVERSITAT POLITÈCNICA DE CATALUNYA
MINISTERIO DE EDUCACIÓN Y CIENCIA
UNIVERSITY OF ALBERTA
IBM