GPI – Global Address Space Programming Interface

GPI is the API that brings PGAS into your standard language

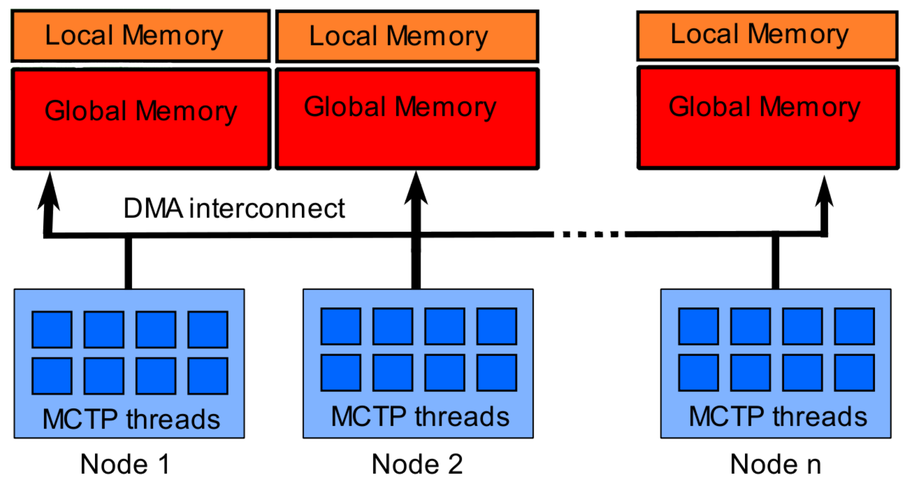
www.gpi-site.com

GPI

Efficient Scalable Multicore
Superior to MPI in performance and scalability

Booth 4347!

- Easy to use: single programming model
- Fast: wirespeed, zero-copy
- Scalable: Perfect overlap, no cycles for communication
- Robust: Fault tolerance
- MCTP: Full hardware awareness (NUMA-Barrier)
- industry-quality, proven in industry applications
- GPI is not a new language, it is an API for standard languages



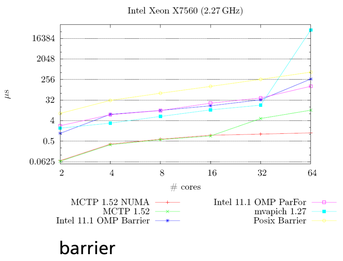
GPI Functionality:

- read and write global data, one-sided, non-blocking, asynchron
- send and receive messages, passive mode
- global atomic counters
- barrier, allReduce
- ranks
- fault tolerant
- daemon concept, environment checks
- auto detect and auto select network

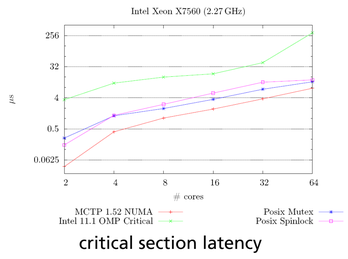
MCTP – Multicore thread package

- supplements GPI on the node level
- fast, very thin abstraction layer
- threadpool fabric, pool synchronization & communication
- full thread control (init, start, stop, wait, exit)
- full system control wrt. HW/Core mappings (logical, physical, SSE level, NUMA layouts, cache affinity)
- thread affinity, get/set mask
- atomic operations fetchAdd, cmpSwap
- fast lock/unlock
- active/passive synchronization, mixable, NUMA-Barrier
- aligned thread safe alloc/free, special pages for global variables
- high resolution timer

Benchmarks:



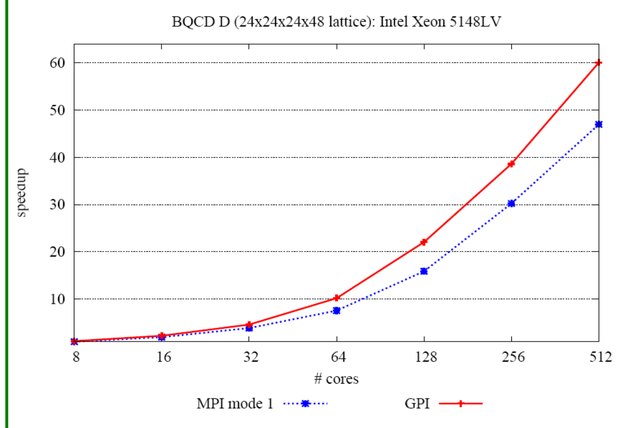
barrier



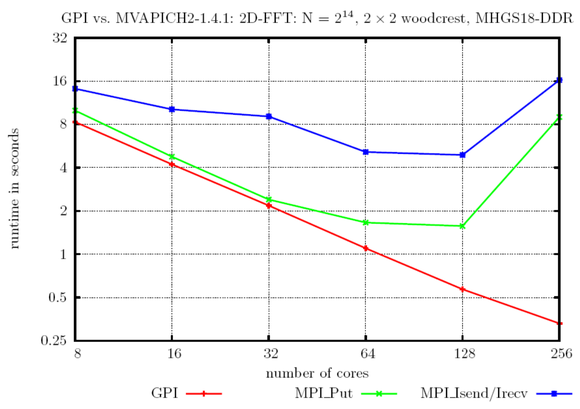
critical section latency

GPI Benchmarks

Quantum chromodynamics: 30% better than MPI



FFT: Enables scaling where MPI fails badly



100% overlap of computation and communication
No waste of CPU cycles!

Industry quotes:

Åre Osen from Statoil AS says
"Our parallel seismic imaging code is based on GPI, since it requires random access to distributed data. We are really satisfied with the performance, the multicore scalability and the robustness of GPI."

Christian Simmendinger from T-Systems means
"For the scalable parallel version of DLR's TAU code we used GPI since MPI did not deliver the required scalability on our large multicore cluster."

Example code: Alltoall in C

```
#include <GPI.h>
#include <GpiLogger.h>
#include <stdio.h>

void dump (const int * arr, const int iProc, const int nProc, const char * msg)
{
  gpi_printf ("%s %i:", msg, iProc);
  for (int i = 0; i < nProc; ++i) gpi_printf (" %2i", arr[i]);
  gpi_printf ("\n");
}

int main (int argc, char *argv[])
{
  startGPI (argc, argv, "", (1UL << 30));       // 1 GiB DMA enabled memory per node

  const int iProc = getRankGPI ();
  const int nProc = getNodeCountGPI ();

  int *mem = (int *) getDmaMemPtrGPI ();        // begin of DMA enabled memory
  int *src = mem;                               // offset 0
  int *dst = mem + nProc;                       // offset nProc * sizeof(int)

  for (int p = 0; p < nProc; ++p)
    {
      src[p] = iProc * nProc + p;

      const unsigned long locOff = p * sizeof (int);
      const unsigned long remOff = (nProc + iProc) * sizeof (int);

      writeDmaGPI (locOff, remOff, sizeof (int), p, GPIQueue0);
    }

  waitDmaGPI (GPIQueue0);
  barrierGPI ();

  dump (src, iProc, nProc, "src");
  dump (dst, iProc, nProc, "dst");

  shutdownGPI ();
}

/* > gcc --std=c99 alltoall.c -I $GPI_HOME/include -L $GPI_HOME/lib64 -lGPI -libverbs15
 * > cp a.out $GPI_HOME/bin
 * > getnode -n 4
 * > $GPI_HOME/bin/a.out
 * [...collected and sorted output...]
 * src 0:  0  1  2  3   dst 0:  0  4  8 12
 * src 1:  4  5  6  7   dst 1:  1  5  9 13
 * src 2:  8  9 10 11   dst 2:  2  6 10 14
 * src 3: 12 13 14 15   dst 3:  3  7 11 15 */
```

For GPI application demonstration:
Visit Fraunhofer Booth 4347!

http://www.gpi-site.com

Fraunhofer ITWM