GPI - Global Address Space Programming Interface

Model, Experiences and Scalability, Future

14th Teraflop Workshop
Stuttgart, December 5th 2011

Mirko Rahn
Fraunhofer ITWM
Competence Center for HPC and Visualization
Fraunhofer ITWM
Competence Center HPC and Visualization

Applications
- PSPRO
- GRT

Tools
- FhGFS
- GPI
- MCTP

Frameworks
- GraPA
- SDPA

Scalability (128 Clients)
GPI - Global address space programming interface

- read/write global data, one-sided, non-blocking, asynchronous, zero-copy
- send/recv messages, passive
- barrier, allReduce
- atomic counters
- ranks
- queues
- fault tolerance

- daemon concept
- NUMA support
- auto detect and auto select network
GPI – Example code: Alltoall in C

```c
#include <GPI.h>

extern void dump (const char *, const int *);

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", src);
    dump ("dst", dst);
    shutdownGPI ();
}
```

```bash
> gcc alltoall.c -I $GPI_HOME/include -L $GPI_HOME/lib64 -lGPI -libverbs15 -o $GPI_HOME/bin/alltoall

> getnode -n 4
> $GPI_HOME/bin/alltoall

[...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
```
GPI - Application examples

- **TAU:** Computational fluid dynamics (DLR, T-Systems SfR), classical MPI decomposition
  - 3D Finite Volume Reynolds Averaged Navier Stokes Solver
  - “key enabler for meeting strategic goals of future air transportation”

- **BQCD:** theory of strong interaction (Nakamura, Stüben), classical Stencil code
  - prediction of particle masses
  - Fortran90, MPI/OMP hybrid and shmem

- **UTS:** Unstructured tree search benchmark (Ohio State et. al.), generic parallel graph algorithm, different implementations available
  - dynamically generated search space
  - static scheduling is unlikely to work well
GPI – BQCD with GPI (D. Grünewald)

- Naïve conversion: Write shmem interface for GPI (no overlap)

- A simple substitution of MPI-calls by GPI-calls is not the solution!
GPI - BQCD with GPI and overlapping

- Overlap communication and computation, *not GPI specific!*

**MPI:**
- Isend, Irecv,
- Test/Waitsome

**GPI:**
- writeDMA,
- synchronization protocoll

---

**BQCD (24 × 24 × 24 × 48): covariant derivate D, overlap efficiency**

(HLRS, Xeon E5440, 16 GiB, DDR Infiniband)

![Graph showing overlap efficiency vs. number of cores](image)

- Overlap efficiency $\eta = \frac{T_{\text{comp.}}}{T_{\text{tot}}}$
- $T_{\text{comm.}} > T_{\text{comp.}}$
- #cores: 16, 32, 64, 128, 256, 512
GPI - BQCD with GPI and overlapping, speedup

BQCD \((24 \times 24 \times 24 \times 48)\): speedup
(HLRS, Xeon E5440, 16 GiB, DD Infiniband)

- MPI/OMP nonoverlapping
- MPI/OMP overlapping
- GPI/OMP overlapping
- ideal scaling

<table>
<thead>
<tr>
<th></th>
<th>speedup @512 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI/OMP no overlap</td>
<td>30.24</td>
</tr>
<tr>
<td>MPI/OMP overlap</td>
<td>42.77</td>
</tr>
<tr>
<td>GPI/OMP overlap</td>
<td>48.37</td>
</tr>
</tbody>
</table>

Speedup normalized to 16 core MPI/OMP nonoverlapping performance.
TAU - Scalability on all levels (C. Simmendinger)

- Core (Instruction Level): SSE2, SSE4.2, AVX, MIC
  - SIMD optimization done by a source to source translator
- Thread (Task Level): generic model beyond OpenMP's fork/join
  - Mutual exclusion and mutual completion in unstructured meshes
  - Fully asynchronous, replace synchronization by local data dependencies
- Communication (System Level)
  - Hide communication using the GPI/PGAS approach

Overall goals: Maintainability, Programmability, Performance Portability
TAU – Strong Scaling:

C. Simmendinger: “Achieving $O(\text{Petascale})$ within the constraints of a typical production run seems feasible.”
UTS – Exploit GPI/PGAS features (R. Machado)

- Work stealing scheme
  - working
  - stealing
  - termination detection
- 2 Levels: Intra node and inter node

- Memory is globally visible: store work queue there
- Prefetch while calculating: communication hidden
- Look at remote state before asking for work: avoid useless communication
- Asynchronous communication: answer questions while calculating
UTS – Best performance values ever

UTS (geometric tree of $270 \times 10^9$ nodes): performance
(Xeon X5670, 16 GiB, QDR Infiniband)

- MVAPICH2 1.5.1
- GPI

2.5x over MPI
89% relative efficiency
7.12 relative speedup
UTS - Best performance values ever

UTS (geometric tree of $270 \times 10^9$ nodes): performance
(Xeon X5670, 16 GiB, QDR Infiniband)

MPI does not benefit from more cores!
GPI - Summary & Future

GPI: powerful tool for the development of scalable parallel applications

- fast: wirespeed, zero-copy
- scalable: perfect overlap, no cycles for communication
- easy to use: (with MPI experience) easy and intuitive to program
- single programming model: no hybrid implementation anymore

There is no free lunch:

- algorithmic changes might be required to achieve asynchronicity
- these changes have usually nothing to do with GPI itself

GASPI: (BMBF project with FhG, T-Systems, FZJ, KIT, DLR, TUD, DWD, scapos)

- define standard for PGAS-API, efficient open source implementation
- Vampir support, algebra libraries, ports of industry codes, training