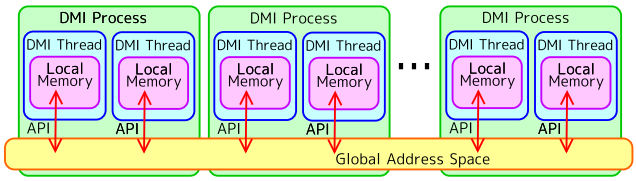
A Global Address Space Framework for Irregular Applications

Kentaro Hara and Kenjiro Taura (The University of Tokyo)

The Goal

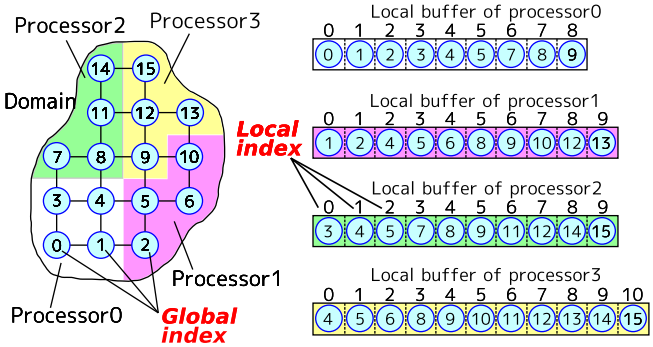
- Goal: ***"Make it easier to develop real-world scientific computings with irregular domain decompositions"***
- Proposal: ***DMI*** (Distributed Memory Interface)
  - → A parallel and distributed programming framework based on a ***global address space*** model

What is DMI?



- A multi-threaded global address space framework for HPC applications
  - → A shared library for C
- Basic APIs:
  - → ***addr = DMI_mmap(size, num)***
    - ♦ Allocate the global address space with *num* pages of (not OS page size but ***arbitrary***) *size* bytes in size
    - ♦ Page-based consistency
  - → ***DMI_read(addr, size, buf)*** , ***DMI_write(addr, size, buf)***
    - ♦ Read/Write *size* bytes from/to the global address space *addr* to/from a local memory *buf*
- Other features (beyond this poster's scope):
  - → Support for dynamic joining and leaving of nodes
  - → Live thread migration for automatic load balancing
  - → APIs for optimizing data locality explicitly

What is the Problem?



- Application: ***Real-world*** Finite Element Methods (FEM)
  - → An ***irregular*** domain decomposition and ordering
  - → Titanium[1], Global-Arrays[2] and XcalableMP support only regular domain decompositions
- Algorithm: Repeat iterations until convergence
  - (1) Obtain the values of ghost points
  - (2) Store the values of interior points and the ghost points in a local buffer according to some suitable ordering
  - (3) Update the values of the interior points using a given connectivity
- Programming bottleneck: ***Exchanging the values of the ghost points***
  - → A programmer has to calculate ***the correspondence between global indexes and local indexes*** :
    - (1) the local indexes of the values that should be sent/received to/from the neighboring processors
    - (2) the processors to/from which these values should be sent/received
  - → ***Local-view programming is too complicated!!!***

An FEM program using DMI

- ***Productive global-view programming***

```
void fem_code_of_processor_i (int i /* a processor i */
    , int n /* the number of processors */
    , int64_t d_addr /* a domain to be decomposed */ ) {
  int rn, wn, iter, u, v;
  int *ri, *rbuf, *wi, *wbuf;
  matrix_t *mat;
  DMI_rwset_t rwset;

  mat = the matrix of the processor i in a CRS format
  wn = the size of the write set of the processor i;
  wi = malloc(sizeof(int) * wn);
  wbuf = malloc(sizeof(double) * wn);
  if(i == 0) wi[0..wn-1] = {0,1,3,4};
  else if(i == 1) wi[0..wn-1] = {2,5,6,10};
  else if(i == 2) wi[0..wn-1] = {7,8,11,14};
  else if(i == 3) wi[0..wn-1] = {9,12,13,15};
  rn = the size of the read set of the processor i;
  ri = malloc(sizeof(int) * rn);
  rbuf = malloc(sizeof(double) * rn);
  if(i == 0) ri[0..rn-1] = {0,1,2,3,4,5,7,8,9};
  else if(i == 1) ri[0..rn-1] = {1,2,4,5,6,8,9,10,12,13};
  else if(i == 2) ri[0..rn-1] = {3,4,5,7,8,9,11,12,14,15};
  else if(i == 3) ri[0..rn-1] = {4,5,6,8,9,10,11,12,13,14,15};
  for(u = 0; u < wn; u++)
    wbuf[u] = 0.00;  /* define initial values of the write set */
  DMI_rwset_decompose(d_addr, i, wi, wn, ri, rn);
  barrier between n processors;
  DMI_rwset_alloc(&rwset, d_addr, i);

  for(iter = 0; /* until convergence */; iter++) {
    ...; /* the CG method */
    barrier between n processors;
    DMI_rwset_write(&rwset, wbuf);
    barrier between n processors;
    DMI_rwset_read(&rwset, rbuf);
    for(u = 0; u < wn; u++) {
      wbuf[u] = 0;
      for(v = mat->row[u]; v < mat->row[u + 1]; v++) {
        wbuf[u] += mat->val[v] * rbuf[mat->col[v]];
      }
    }
    ...; /* the CG method */
  }
  DMI_rwset_free(&rwset);
}
```

Define a write set using global indexes

Define a read set using local indexes

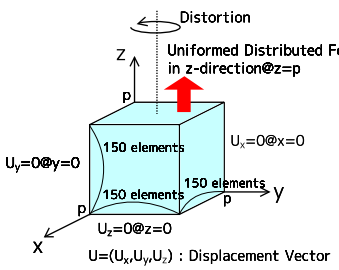
Decompose a domain

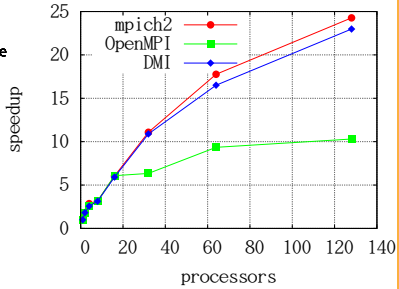
Write the values of the write set

Read the values of the read set

- Internal implementations:
  - → Manage point values efficiently in a manner similar to local-view programming by ***internally transforming the global indexes specified by a programmer into the local indexes***
  - → ***Aggregate multiple internal communications*** for the same processor into one message

Performance Evaluation





- Environment: 8 cores×16 nodes, 10 Gbit Ethernet
- Experiments:
  - → Stress analysis using an FEM
  - → A ***real-world*** and ***hard-to-converge*** problem
  - → DMI vs the champion MPI program of the parallel programming contest
- Results:
  - → DMI is ***easier*** to program than MPI
  - → Performance: ***mpich2≈DMI≫OpenMPI***

References

[1] Jimmy Su, Tong Wen, and Katherine Yelick. Compiler and Runtime Support for Scaling Adaptive Mesh Refinement Computations in Titanium. Technical report, Electrical Engineering and Computer Sciences University of California at Berkeley, Jun 2006.

[2] Jarek Nieplocha, Bruce Palmer, Vinod Tipparaju, Manojkumar Krishnan, Harold Trease, and Edo Apra. Advances, Applications and Performance of the Global Arrays Shared Memory Programming Toolkit. *International Journal of High Performance Computing Applications*, Vol. 20, No. 2, pp. 203–231, 2006.