Towards next generation parallel language framework for Petascale systems: 
XcalableMP project and Experience from HPF

Mitsuhisa Sato
University of Tsukuba
Agenda

- Lesson learned from HPF
  - Think about MPI …
  - History of HPF in Japan

- XcalableMP: directive-based language eXtension for Scalable and performance-tunable Parallel Programming
  - Motivation
  - Concept and model
  - Some examples
Message Passing Model (MPI)

- Message passing model was the dominant programming model in the past.
  - …. Yes.

- Message passing is the dominant programming model today.
  - … Unfortunately, yes…

- Will OpenMP be a programming model for future system?
  - OpenMP is only for shared memory model.

- Are programmers satisfied with MPI?
  - yes…? Many programmers writes MPI.

- Is MPI enough for parallelizing scientific parallel programs?

- Application programmer’s concern is to get their answers faster!!
  - Automatic parallelizing compiler is the best, but …
    many problems remain.

- Why was MPI accepted and so successful?
  - Portability and Education, and more …?
The rise and fall of High Performance Fortran in Japan
~Lessons learned from HPF ~
(by Sakagami@NIFS and Murai@NEC)

- (A similar retrospective paper was published by Prof. Ken Kennedy and Zima)

- Background of HPF (in 1992-1997, 1st draft)
  - MPI (message passing model) was (still now) an obstacle for programming distributed memory systems.
    - Debugging MPI code is not easy, and update/modification of MPI program forces a tough work for application people.
    - If MPI is only a solution to parallel machine, nobody wants to use parallel machines. (EP is ok, but …)
  - There was a great demand for parallel programming languages!
    - Application people want just easy parallel programming environment with reasonable (not necessarily perfect) performance.
    - OpenMP is just for shared memory systems.
    - Not practical alternative solutions. (Now, how about HPCS languages?!)
HPF history in Japan

- Japanese supercomputer vendors were interested in HPF and developed HPF compiler on their systems.
- NEC has been supporting HPF for Earth Simulator System.
- Japan HPF promotion consortium was organized by NEC, Hitachi, Fujitsu …
  - HPF/JA proposal
- Still survive in Japan, supported by Japan HPF promotion consortium
- Compiler Availability
  - HPF/ES (HPF+HPF/JA+some extension for Earth Simulator)
  - HPF/SX, HPF/VPP, HPF/ES for PC clusters, fhpf (free software distributed by HPF consortium)
“Pitfalls” and Lessons learned from HPF (1)

- “Ideal” design policy of HPF
  - A user gives a small information such as data distribution and parallelism.
  - The compiler generates “good” communication and work-sharing automatically.
  - By ignoring directives, parallelized code can be considered as the original sequential code.
  - Large specifications were included to satisfy “theoretical” completeness of the language model.

- Lesson: “*Don’t give too much expectation to users which the technology could not meet.*”
  - This “ideal” design policy had generated a great “expectation” from users! But, the reality was not …
  - Initial (reference) implementation is important to attract people.
    - No reference implementation of HPF like MPICH in MPI standard.
“Pitfalls” and Lessons learned from HPF (2)

- The base language of HPF was “immature” F90
  - A bad thing was that at the moment of HPF announced (mid 90’s), F90 was still immature.
  - Many application people had to rewrite programs in F90 in order to use HPF
    - Re-write from F77 to F90 was not easy work.
  - No C/C++

- **Lesson**: “Application people don’t want to rewrite their programs. They are very conservative”
  - Sometimes, they complained that “I re-wrote my program by spending a lot time, but the performance was not good!”
  - The reason why the performance of HPF was not so good was sometimes due to the immaturity of F90 implementation.
“Pitfalls” and Lessons learned from HPF (3)

- No explicit mean for performance tuning.
  - Everything depends on compiler optimization.
  - Users can specify more detail directives, but no information how much performance improvement will be obtained by additional informations
    - INDEPENDENT for parallel loop
    - PROCESSOR + DISTRIBUTE
    - ON HOME
  - The performance is too much dependent on the compiler quality, resulting in “incompatibility” due to compilers.

- Lesson: “Specification must be clear. Programmers want to know what happens by giving directives”
  - The way for tuning performance should be provided.
“Petascale” Parallel language design working group

- **Objectives**
  - Making a draft on “petascale” parallel language for “standard” parallel programming
  - To propose the draft to “world-wide” community as “standard”

- **Members**
  - Academia: M. Sato, T. Boku (compiler and system, U. Tsukuba), K. Nakajima (app. and programming, U. Tokyo), Nanri (system, Kyusyu U.), Okabe (HPF, Kyoto U.)
  - Research Lab.: Watanabe and Yokokawa (RIKEN), Sakagami (app. and HPF, NIFS), Matsuo (app., JAXA), Uehara (app., JAMSTEC/ES)
  - Industries: Iwashita and Hotta (HPF and XPFortran, Fujitsu), Murai and Seo (HPF, NEC), Anzaki and Negishi (Hitachi)

- **More than 10 WG meetings have been held (Dec. 13/2007 for kick-off)**

- **Funding for development**
  - E-science project: “Seamless and Highly-productive Parallel Programming Environment for High-performance computing” project funded by Ministry of Education, Culture, Sports, Science and Technology, JAPAN.
    - Project PI: Yutaka Ishiakwa, co-PI: Sato and Nakashima(Kyoto), PO: Prof. Oyanagi
    - Project Period: 2008/Oct to 2012/Mar (3.5 years)
Requirements of “petascale” language

- **Performance**
  - The user can achieve performance “equivalent to in MPI”
  - More than MPI – one-sided communication (remote memory copy)

- **Expressiveness**
  - The user can express parallelism “equivalent in MPI” in easier way.
  - Task parallelism – for multi-physics

- **Optimizability**
  - Structured description of parallelism for analysis and optimization
  - Should have some mechanism to map to hardware network topology

- **Education cost**
  - For non-CS people, it should be not necessarily new, but practical
XcalableMP: directive-based language eXtension for Scalable and performance-tunable Parallel Programming

- **Directive-based language extensions** for familiar languages F90/C/C++
  - To reduce code-rewriting and educational costs.

- **“Scalable” for Distributed Memory Programming**
  - SPMD as a basic execution model
  - A thread starts execution in each node independently (as in MPI).
  - Duplicated execution if no directive specified.
  - MIMD for Task parallelism

- **“performance tunable” for explicit communication and synchronization.**
  - Work-sharing and communication occurs when directives are encountered.
  - All actions are taken by directives for being “easy-to-understand” in performance tuning (different from HPF).
Overview of XcalableMP

- XMP supports typical parallelization based on the **data parallel paradigm** and work sharing under "**global view**"
  - An original sequential code can be parallelized with **directives**, like OpenMP.
- XMP also includes CAF-like PGAS (Partitioned Global Address Space) feature as "**local view**" programming.

![Diagram of XMP parallel execution model]

**User applications**

- **Global view Directives**
  - Support common pattern (communication and work-sharing) for data parallel programming
  - Reduction and scatter/gather
  - Communication of sleeve area
  - Like OpenMPD, HPF/JA, XFP

- **Local view Directives (CAF/PGAS)**

**XMP parallel execution model**

- Two-sided comm. (MPI)
- One-sided comm. (remote memory access)

**MPI Interface**

**XMP runtime libraries**
```c
int array[YMAX][XMAX];

#pragma xmp nodes p(4)
#pragma xmp template t(YMAX)
#pragma xmp distribute t(block) on p
#pragma xmp align array[i][*] to t(i)

main(){
    int i, j, res;
    res = 0;

    #pragma xmp loop on t(i)  reduction(+:res)
    for(i = 0; i < 10; i++)
        for(j = 0; j < 10; j++){
            array[i][j] = func(i, j);
            res += array[i][j];
        }
}
```

**data distribution**

**add to the serial code: incremental parallelization**

**work sharing and data synchronization**
The same code written in MPI

```c
int array[YMAX][XMAX];

main(int argc, char**argv){
    int i,j,res,temp_res, dx,llimit,ulimit,size,rank;

    MPI_Init(argc, argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    dx = YMAX/size;
    llimit = rank * dx;
    if(rank != (size - 1)) ulimit = llimit + dx;
    else ulimit = YMAX;

    temp_res = 0;
    for(i = llimit; i < ulimit; i++)
        for(j = 0; j < 10; j++){
            array[i][j] = func(i, j);
            temp_res += array[i][j];
        }

    MPI_Allreduce(&temp_res, &res, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
    MPI_Finalize();
}
```
Nodes, templates and data/loop distributions

- Idea inherited from HPF
- Node is an abstraction of processor and memory in distributed memory environment.

```
#pragma xmp nodes p(32)
```

- Template is used as a dummy array distributed on nodes

```
#pragma xmp template t(100)
#pragma xmp distribute t(block) on p
```

- A global data is aligned to the template

```
#pragma xmp distribute array[i][*] to t(i)
```

- Loop iteration must also be aligned to the template by on-clause.

```
#pragma xmp loop on t(i)
```
Array data distribution

- The following directives specify a data distribution among nodes:
  - `#pragma xmp nodes p(*)`
  - `#pragma xmp template T(0:15)`
  - `#pragma xmp distribute T(block) on p`
  - `#pragma xmp align array[i] to T(i)`

![Diagram showing data distribution among nodes]

Array distribution:

- `array[]`
- `node0` (red)
- `node1` (blue)
- `node2` (yellow)
- `node3` (green)

Reference to assigned to other nodes may cause errors!!

Assign loop iteration as to compute own regions

Communicate data between other nodes
Parallel Execution of “for” loop

- Execute for loop to compute on array

```verbatim
#pragma xmp loop on t(i)
for(i=2; i <= 10; i++)
```

Data region to be computed by for loop

Execute “for” loop in parallel with affinity to array distribution by on-clause:

```verbatim
#pragma xmp loop on t(i)
```

Array distribution
Data synchronization of array (shadow)

- Exchange data only on “shadow” (sleeve) region
  - If neighbor data is required to communicate, then only sleeve area can be considered.
  - Example: \( b[i] = \text{array}[i-1] + \text{array}[i+1] \)
    
    ```
    #pragma xmp align array[i] to t(i)
    ```

    ![Diagram showing array synchronization]

    #pragma xmp shadow array[1:1]

    ![Diagram showing node areas]

    **Programmer specifies sleeve region explicitly**

    **Directive**: `#pragma xmp reflect array`
XcalableMP example (Laplace, global view)

```c
#pragma xmp nodes p(NPROCS)
#pragma xmp template t(1:N)
#pragma xmp distribute t(block) on p

double u[XSIZE+2][YSIZE+2],
    uu[XSIZE+2][YSIZE+2];
#pragma xmp align u[i][*] to t(i)
#pragma xmp align uu[i][*] to t(i)
#pragma xmp shadow uu[1:1][0:0]

for(k = 0; k < NITER; k++){
    /* old <- new */
    #pragma xmp loop on t(x)
    for(x = 1; x <= XSIZE; x++)
        lap_main()
    {
        int x,y,k;
        double sum;
        ...
        #pragma xmp reflect uu
        #pragma xmp loop on t(x)
        for(x = 1; x <= XSIZE; x++)
            for(y = 1; y <= YSIZE; y++)
                uu[x][y] = u[x][y];
        #pragma xmp block on master
        printf("sum = %g\n",sum);
    }
    /* check sum */
    sum = 0.0;
    #pragma xmp loop on t[x] reduction(+:sum)
    for(x = 1; x <= XSIZE; x++)
        for(y = 1; y <= YSIZE; y++)
            sum += (uu[x][y]-u[x][y]);
    #pragma xmp block on master
    printf("sum = %g\n",sum);
}
```

Use "align" to specify data distribution
For data synchronization, use "shadow" directive
specify sleeve area

Loop partitioning
And scheduling

Data synchronization

Definition of nodes

Template to define distribution
Data synchronization of array (full shadow)

- Full shadow specifies whole data replicated in all nodes
  - `#pragma xmp shadow array[*]`
- Reflect operation to distribute data to every nodes
  - `#pragma reflect array`
  - Execute communication to get data assigned to other nodes
  - Most easy way to synchronize → But, communication is expensive!

```
array[]
```
```
node0
```
```
node1
```
```
node2
```
```
node3
```

Now, we can access correct data by local access!!
XcalableMP example (NPB CG, global view)

```c
/* code fragment from conj_grad in NPB CG */
sum = 0.0;
#pragma xmp loop on t(j) reduction(+:sum)
for (j = 1; j <= lastcol-firstcol+1; j++) {
    sum = sum + r[j]*r[j];
}
rho = sum;
for (cgit = 1; cgit <= cgitmax; cgit++) {
    #pragma xmp reflect p
    #pragma xmp loop on t(j)
    for (j = 1; j <= lastrow-firstrow+1; j++) {
        sum = 0.0;
        for (k = rowstr[j]; k <= rowstr[j+1]-1; k++)
            sum = sum + a[k]*p[colidx[k]];
        w[j] = sum;
    }
}
#pragma xmp loop on t(j)
for (j = 1; j <= lastcol-firstcol+1; j++) {
    q[j] = w[j];
}```
XcalableMP Global view directives

- Execution only master node
  - #pragma xmp block on master

- Broadcast from master node
  - #pragma xmp bcast (var)

- Barrier/Reduction
  - #pragma xmp reduction (op: var)
  - #pragma xmp barrier

- Global data move directives for collective comm./get/put

- Task parallelism
  - #pragma xmp task on node-set
XcalableMP Local view directives

- XcalableMP also includes CAF-like PGAS (Partitioned Global Address Space) feature as "local view" programming.
  - The basic execution model of XcalableMP is SPMD
    - Each node executes the program independently on local data if no directive
  - We adopt Co-Array as our PGAS feature.
  - In C language, we propose array section construct.
  - Can be useful to optimize the communication
  - Support alias Global view to Local view

Array section in C

```c
int A[10]:
int B[5];
A[4:9] = B[0:4];
```

- For flexibility and extensibility, the execution model allows combining with explicit MPI coding for more complicated and tuned parallel codes & libraries.
  - Need to interface to MPI at low level to allows the programmer to use MPI for optimization
  - It can be useful to program for large-scale parallel machine.

- For multi-core and SMP clusters, OpenMP directives can be combined into XcalableMP for thread programming inside each node for hybrid programming.
Position of XcalableMP

Degree of Performance tuning

Cost to obtain Performance

Automatic parallelization

Programming cost

XcalableMP

PGAS

MPI

chapel

HPF
Our objective of “language working group” is to design “standard” parallel programming language for petascale distributed memory systems

- High productivity for distributed memory parallel programming
- Not just for research, but collecting ideas for “standard”
- Distributed memory programming “better than MPI” !!!

XcalableMP project: status and schedule

- 1Q/09 first draft of XcalableMP specification
- 2Q/09 β release, C language version
- 3Q/09 Fortran version (for SC09 HPC Challenge!)
- Ask international community for review of the specification
Thank you for your attention!!!

XcalableMP is under design. Any comments and contributions will be very welcome!

Q & A?