言語に依存しないXMP APIによるプログラミングモデル

佐藤　三久

Team Leader, Programming Environment Research Team, RIKEN CCS

Professor (Cooperative Graduate School Program), University of Tsukuba

XMP APIについて

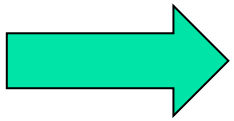
- 言語拡張・コンパイラアプローチでは限界があるの で、XMPとなるべく「同等」なプログラミングができる、ライブラリAPIを定義することを提案したい

- 基本的な方針・イメージ
  - XMPの基本的なオブジェクトである、ノード集合、テンプレート、分散配列等を、ディスクリプタで表現し、それに対するdirectiveの操作をAPIで表現する。

- APIによる展開
  - C++ template versionでの利用で考えてみる。
  - 結局、XMPから言語的なsugar? をのぞいたら、何が残るのかに興味がある。
  - 逆に、言語的なアプローチの重要性が明らかになる？

“Compiler-free” approaches for PGAS

- Approach by compiler will give:
  - New language, or language extension provides easy-to-use and intuitive feature resulting in better productivity.
  - Enable compiler analysis for further optimization: removal of redundant sync and selection of efficient communication, etc, ...
  - But, in reality, compiler-approach is not easy to be accepted for deployment, and support many sites, ...
- “Compiler-free” approach
  - Library approach: MPI3 RMA, OpenShmem, GlobalArray, ...
  - C++ Template approach: UPC++, DASH, ...
- This approach may increase portability, clean separation from base compiler optimization



XMP API

「裏事情」

- 理研のプログラミング環境研究チームの評価委員会で、XMPの実際のアプリに適用できるようにするべき、というコメントがあった。

- この言語的なアプローチは必ずしもアプリケーションプログラマに受けいられていない現状がある。これはコンパイラや関連ソフトウエアの完成度だけの問題ではなく、いろいろなプラットフォームに対応するための可搬性の問題や、さらには新しいソフトウエアへの心理的な障害といった問題もある。
  - 例えば、いったん新しい言語を記述したプログラムを他のプラットフォームにもっていくためにはサポートされていることが必要であるが、新しい言語では必ずしもサポートされるとは限らない。
- そのために、…

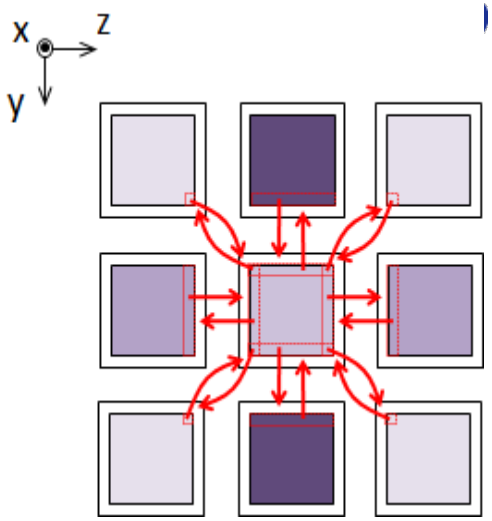
何がメリットになるのか

- “MPI is too low and too high API for communication”. (Prof. Marc Snir, JLESC 7th WS)
  - MPI RMA APIs offer their PGAS model rather than “primitives” for other PGAS.
- **MPIよりも低レベルのリモートメモリ通信の活用**
  - 現在、IB Verbsを直接つかった通信レイヤを検討
  - Fugakuでは、u-Tofuを使った通信レイヤを実装中

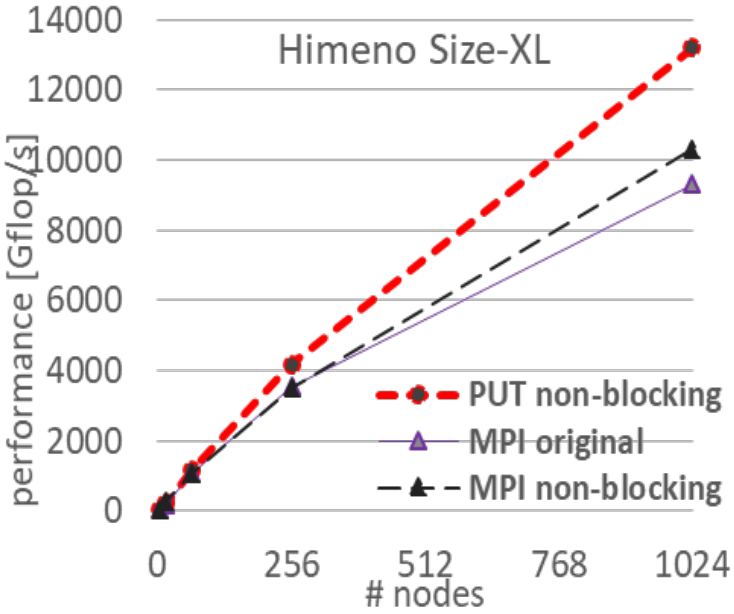
- Advantages of RMA/RDMA Operations
  - multiple data transfers can be performed with a single synchronization operation
  - Some irregular communication patterns can be more economically expressed
  - Significantly faster than send/receive on systems with hardware support for remote memory access
    - Recently, many kinds of high-speed interconnect have hardware support for RDMA, including Infiniband, ... as well as Cray and Fujitsu.

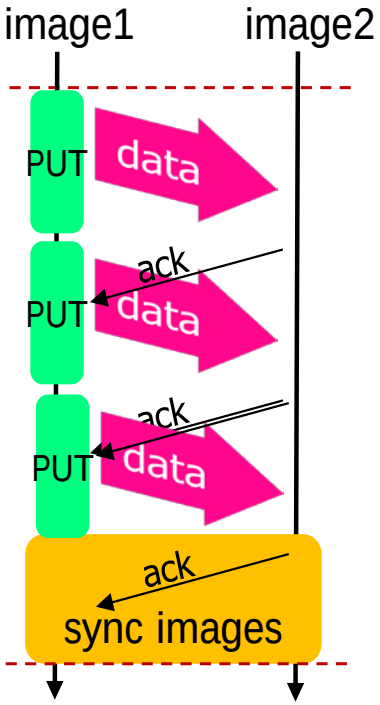
Optimized Comm. by RDMA: stencil communication

- Typical communication pattern in domain-decomposition.
- Advantage of PGAS: Multiple data transfers with a single synchronization operation at end
- PUT non-blocking outperforms MPI in Himeno Benchmark!
  - Don’t wait ack before sending the next data (by FJ-RDMA)



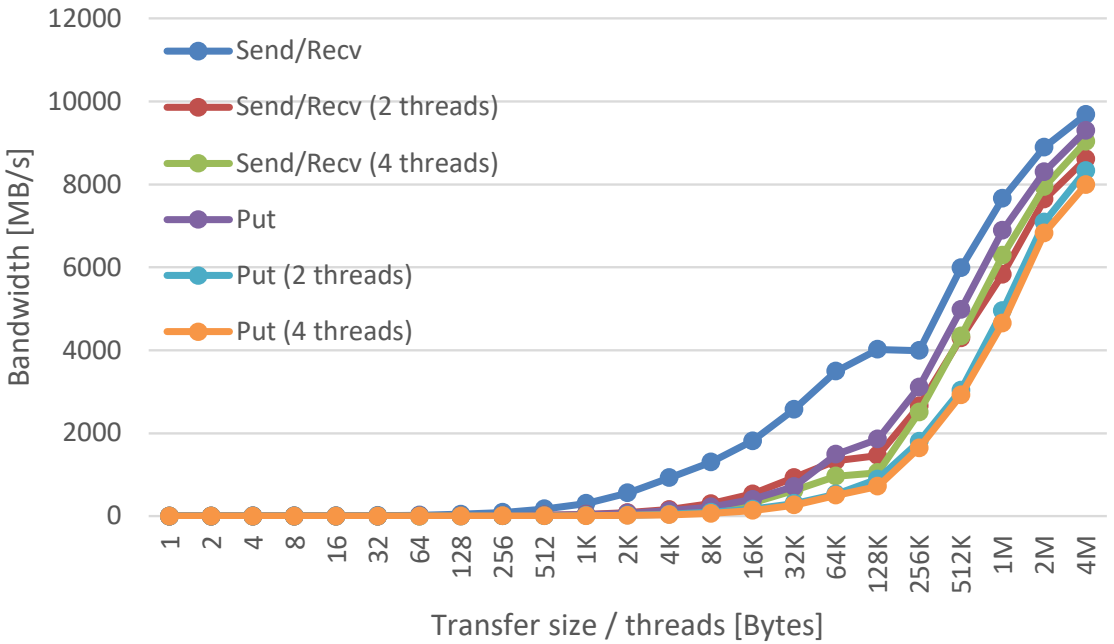
NOTE: The detail of this results is to be presented in HPCAisa 2018: Hidetoshi Iwashita, Masahiro Nakao, Hitoshi Murai, Mitsuhisa Sato, “A Source-to-Source Translation of Coarray Fortran with MPI for High Performance”





Communication in multithreaded env.
Communication performance on KNL system

- **Aggregate comm. performance between multiple threads between nodes**
  - The performance using “Send/Recv” may be better than that using “Put”.
  - the performance of MPI_THREAD_MULTIPLE is lower than that of the single-threaded communication.

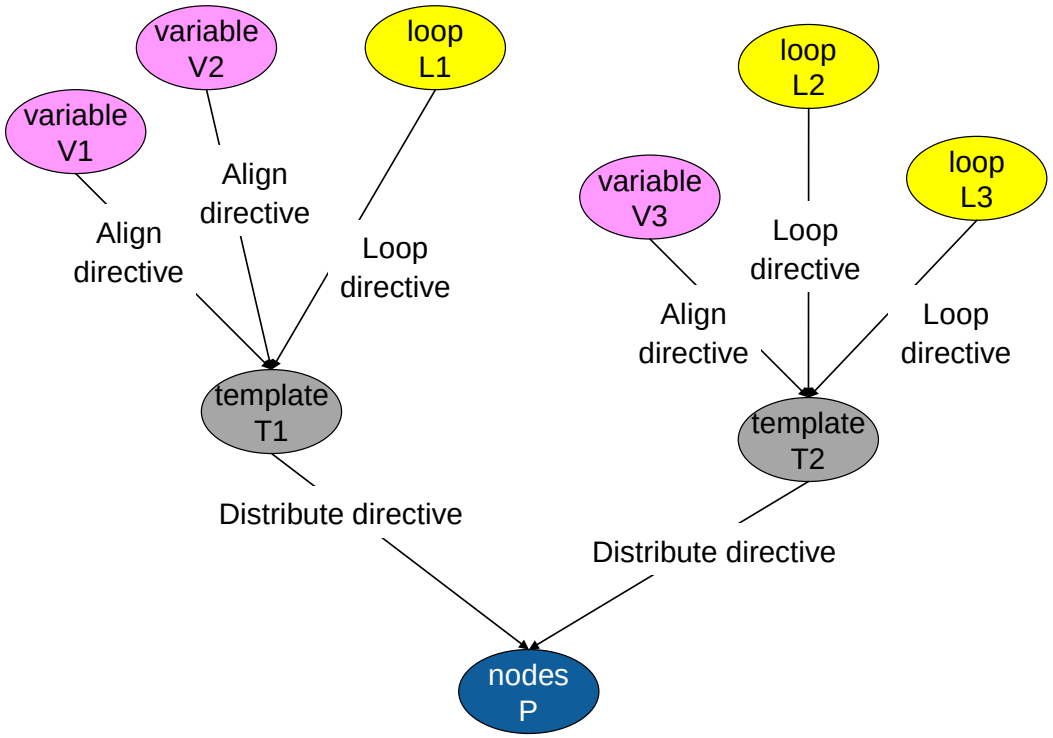


- **Why MPI_THREAD_MULTIPLE is so slow?**
  - “giant-lock” for message ordering and tag matching for wild-card.
  - New proposal is being discussed: “end-point” for thread.
- **Can RDMA be another “light-weight” solution for communication in multithreaded execution of manycore?**

Design of XMP API (1)

- High-level “array-oriented” interface
  - Using the concept of “template” to describe the information of distribution
  - Allocation of global distributed array with several distribution forms
  - Efficient “reflect” operation to update “halo” region
  - Compute mapping indices from local to global, from global to local.

- Node is an abstraction of processor and memory in distributed memory environment, declared by node directive.
- Template is used as a dummy array distributed on nodes
- A global data is aligned to the template
- Loop iteration may also be aligned to the template



XcalableMP Code Example (C)

```
double array[YMAX][XMAX];
double B[10][10];

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

main(){
  int i, j;
  double res = 0.0;
#pragma xmp loop on t(i)  reduction(+:res)
  for(i = 0; i < YMAX; i++)
    for(j = 0; j < XMAX; j++){
      array[i][j] = func(i, j);
      res += array[i][j];
    }
….
#pragma xmp gmove in
  B[0:10][0:10] = A[0:10][0:10];
}
```

data distribution

add to the serial code : incremental parallelization

work mapping and
data synchronization

Proposed API (1)

- Set of nodes: xmp_nodes_t
  - xmp_nodes_t *xmp_new_nodes(int n_dims) // constructor

    ex: xmp_nodes_t *np = xmp_new_nodes(1)
  - xmp_nodes_set_dim_size(xmp_template_t *,int dim_idx, int size) // set size

    ex: xmp_nodes_set_dim_size(np,0,4)
- template: xmp_template_t
  - xmp_template_t *xmp_new_template(int n_dims) // constructor

    ex: xmp_template_t *tp = xmp_new_template(1)
  - xmp_template_set_dim_size(xmp_template_t *, int dim_idx, xmp_size_t low, xmp_size_t high)

    ex: xmp_template_set_dim_size(tp,0,0,YMAX)
  - xmp_distribute_template(xmp_template_t *, int template_dim_idx, xmp_node_t *, int node_dim_idxe, enum dist_kind) // distribute template on nodes

    ex: xmp_distribute_template(tp,0,np,0, XMP_DIST_BLOCK)
- Range : xmp_range_t // To describe range in array
  - xmp_range_t *xmp_new_range(int n_dims) // constructor

    ex: xmp_range_t *ap = xmp_new_range(2)
  - xmp_range_set_dim_size(xmp_array_t *, int dim_idx, xmp_size_t low, xmp_size_t high)

    ex: xmp_range_set_dim_size(ap,0,0,YMAX)

    ex: xmp_range_set_dim_size(ap,1,0,XMAX)

Proposed API (2)

- Global distributed array: xmp_array_t
  - xmp_array_t *xmp_new_array(xmp_range_t *rp) // constructor

    ex: xmp_array_t *ap = xmp_new_array(rp)
  - xmp_align_array(xmp_array_t *, xmp_template_t *, int template_dim_idx, int array_dim_dix, int offset) // align to node

    ex: xmp_align_array(ap,tp,0,1,0)
  - xmp_set_shadow(xmp_array_t *, int dim_idx, int shdw_size_l, int shdw_size_h)
  - xmp_allocate_array(xmp_array_t *, void *local_array) // allocate

    ex: xmp_allocate_array(ap, NULL)
  - void *xmp_get_local_addr(xmp_array_t *) // get local array's address
  - int xmp_get_local_size(ap, int dim_idx)
  - int Xmp_get_local_lda(ap,int dim_idx)

- Index mapping
  - int xmp_l2g(xmp_template_t *tp, int dim_idx, int local_idx) // Local to global
  - int xmp_g2l(xmp_template_t *tp, int dim_idx, int global_idx) // global to local

Proposed API (3)

- Array reflect operation to update shadow

- Simple data transfer: get/put
  - xmp_get(void *local_addr, int node_n, void *remote_addr, xmp_size_t size)
  - xmp_put(void *local_addr, int node_n, void *remote_addr, xmp_size_t size)

- Global Array Data transfer : gmove
  - xmp_gmove(xmp_array_t *src, xmp_range_t *src_rp, xmp_array_t *dst, xmp_range_t *dst_rp) // collective
  - xmp_gmove_in(xmp_array_t *dst, xmp_range_t *dst_rp, xmp_array_t *src, xmp_range_t *rp) // get
  - xmp_gmove_out(xmp_array_t *src, xmp_range_t *src_rp, xmp_array_t *dst, xmp_range_t *dst_rp) // put
  - ... Local memory version

XcalableMP Code Example (C)

```
#define N_PROC 4
double local_array[YMAX/N_PROC][XMAX]; double B[10][10];
main(){
  int i, j;  double res = 0.0;
 xmp_nodes_t *np = xmp_new_nodes(1); // nodes
 xmp_nodes_set_dim_size(np,0,N_PROC);
 xmp_template_t *tp = xmp_new_range(1); // template
 xmp_template_set_dim_size(tp,0,0,YMAX);
 xmp_distribute_template(tp, 0, np, 0, XMP_DIST_BLOCK);
 xmp_range_t *rp = xmp_new_range(2);
 xmp_range_set_dim_size(rp,0,0,YMAX);
 xmp_range_set_dim_size(rp,1,0,XMAX)
 xmp_array_t *ap = xmp_new_array(rp);
 xmp_align_array(ap,0,tp,0,0);
 xmp_allocate_array(ap, local_array);
for(i = 0; i < YMAX/N_PROC; i++)
    for(j = 0; j < XMAX; j++){
      local_array[i][j] = func(xmp_l2G(tp,0,i), j);
      res += local_array[i][j];
    }
 MPI_reduce(… &res, …); // MPI_call
xmp_range_t *rp2 = xmp_new_range(2);
xmp_range_set_dim_size(rp2,0,0,10);xmp_range_set_dim_size(rp2,1,0,10)
xmp_gmove_in_local(B,rp3,ap,rp2);
```

Design of XMP API (2)

- “Low-level” remote memory access interface
  - A thin-layer for hardware RDMA
  - Simple and efficient communication in multithreaded environment
  - Memory-based synchronization

- Array-based data-transfer / remote memory copy
  - Like “Global-Array”, or “Shmem”

  - xmp_get
  - xmp_put
  - Coarrayのallocate

  - Coarray Get/putにおけるArray sectionのサポート
  - Global arrayへのput/getも。

- 初期化・終了
  - xmp_initialize
  - xmp_finalize
- ノードグループ (xmp task directive)の管理
- Intrinsic and Library Procedures、なるべく、そのままの形で使えるようにする。
  - 7.2 System Inquiry Functions
  - 7.3 Execution Control Functions
  - 7.4 Synchronization Functions
  - 7.5 Memory Allocation Functions
  - 7.6 Mapping Inquiry Functions
  - 7.9 Intrinsic/Built-in Transformational Procedures

Tasklet 関係

- Tasklet関係もruntimeのインターフェスを出す。
- データー依存も記述
- 例：Argbotsで実装した例
  - xmp_tasklet_create(void (* Func)(),#arg,void *args,
    #in,void *ins,#out,void *outs);

```
xmp_tasklet_create(task_trsm,4,args,1,in_data,1,out_data);
```

  - void xmp_tasklet_wait()
- これを、ノード間、あるいは演算加速機構（GPU）への拡張も検討中。

Summary and Concluding Remarks

- XMP API includes:
  - High-level “array-oriented” interface inherited from XMP global view
  - “Low-level” remote memory access interface for simple and efficient one-sided comm. in multithreaded environment.
  - Array-based data-transfer / remote memory copy
  - Task-parallelism with data dependency (tasklet)

- XMP API is still under design and prototyping
  - Implementation by making use of current XMP runtime.
  - Also, C++ template wrapping XMP API is planned. (like DASH?)

- Currently, we are also working on low-level communication layer for multithreaded program
  - We expect it must be faster than MPI_THREAD_MULTIPLE
  - A thin-layer of low-level communication layer such as IB Verbs or U-tofu