Scalable Workload Models for System Simulations

John Thompson
August 13, 2014
Background and Motivation

- Intel Omni Scale Fabrics
- There are a few things we need to get right
  - Accurate workload representations
    - Traffic rates
    - Compute/communication overlaps
    - Message sizes
    - Data dependencies
    - Synchronization
    - Load Imbalance
  - Fast and scalable simulations
  - Revealing analytics and visualization
- SWM framework requirements
  - Scalability
  - Composability
  - Extensibility
  - Portability

Figure 3. System simulation framework
SWM Framework

```
{
  "jobs": [
    {
      "name": "aggressor",
      "app": "tornado",
      "size": 3,
      "time": 0,
      "yells": false,
      "cfg": {
        "iteration_cnt": -1,
        "hierarchy_depth": 2,
        "hierarchical_sizes": "3,1",
        "hierarchical_mappings": "right,right",
        "req_traffic": [
          {
            "parts": 1,
            "routing_type": "minimal",
            "bytes": 32,
            "vcs": 0
          }
        ]
      },
      "placement": {
        "type": "offset_stride_weight",
        "offset": 0,
        "stride": 2,
        "weight": 1
      }
    }
  ]
}
```
The SWM Template Library features a variety of calls and support to model:

- MPI (eg: TMPL_Send, TMPL_Isend, TMPL_Alltoall, TMPL_Wait)
- SHMEM (eg: TMPL_Intmalloc, TMPL_Putmem, TMPL_Intadd, TMPL_Fence)
- Synthetics (eg: TMPL_Synthetic)

It is not restricted to the API of a commercial library

- Fine-grained virtual channel configuration is useful in fabric space
- Could explore proposed library extensions (ie: non-blocking collectives)

```cpp
#include "point_to_point_template_user_code.h"

P2PTemplateUserCode::P2PTemplateUserCode(  
    TemplateProcess* process,  
    boost::property_tree::ptree cfg,  
    void**& generic_ptrs,  
    template_topology* topology  
) :  
    AppBaseTemplateUserCode(process, cfg),  
    src_rank_id(cfg.get<uint32_t>("src_rank_id")),  
    dst_rank_id(cfg.get<uint32_t>("dst_rank_id")),  
    compute_delay(cfg.get<uint32_t>("compute_delay",0)),  
    topology(topology)  
{

}

void  
P2PTemplateUserCode::call()  
{
    if (process->process_id == src_rank_id) {
        for(uint32_t iter=0; iter<iteration_cnt; iter++)
        {
            GetReqDetails(  
                req_rt, rsp_rt,  
                req_vc, rsp_vc,  
                msg_req_bytes,msg_rsp_bytes,pkt_rsp_bytes
```
void RingTemplateUserCode::call() {
    uint32_t handle;
    for(uint32_t iter=0; iter<iteration_cnt; iter++) {
        if(process_id == 0) {
            //send to 1
            TMPL_Isend(
                1,       //dst
                0,       //comm
                process_id, //tag
                request_vc,  //reqvc
                response_vc, //rspvc
                0,        //buf
                message_size, //bytes
                &handle,
                req_rt,
                rsp_rt
            );
            TMPL_Wait(handle);
            //recv from process_cnt -1
            TMPL_Irecv(
                process_cnt -1, //src
                0,              //comm
                process_cnt -1, //tag
                0,              //buf
                &handle
            );
            TMPL_Wait(handle);
        }
    }
}
else {
    //recv from process_id -1
    TMPL_Irecv(
        ... 
    );
    TMPL_Wait(handle);
    if(process_id == process_cnt -1) {
        //send to 0
        TMPL_Isend(
            ... 
        );
        TMPL_Wait(handle);
    }
    else {
        //send to process_id +1
        TMPL_Isend(
            ... 
        );
        TMPL_Wait(handle);
    }
}
TMPL_Finalize();
assert(0);
SWM Template Library

- Several SWMs exist today
  - Synthetics (ie: UR, Permutation, Nearest Neighbor, …)
  - Toy (ie: Ring, MDFFT, QCD, …)
  - Application (HACC, Nekbone, and eight others)

Figure 4. Src-Dest traffic matrix – Measured vs. SWM (HACC workload)
Successes, Challenges, To-dos, and Wishes

- **Successes**
  - Enables at-scale system simulation
  - Full-featured -- constructed a comprehensive suite of stimulus
    - MPI, SHMEM, and Synthetic support
    - Many CORAL (DOE) applications have accurate SWM representations now
  - Easily adopted by SWM developers (application experts)
  - Easily interfaced to multiple simulators
  - Allows for the construction of workloads composed of many SWMs

- **Challenges**
  - Requires a deep understanding of application behavior
  - How do you guarantee an SWM’s accuracy, especially at increasing scales?

- **To-dos**
  - Continue to support an ever increase set of template calls
  - Interface review and refinement
  - More flexible meta-data abstraction (I might want to append VC data, you might want something else)

- **Wishes**
  - Wider adoption of some such common framework
  - Obtain SWM models from 3rd parties