The future is asynchronous …

Vivek Sarkar

Professor & Chair, School of Computer Science

Stephen Fleming Chair

College of Computing

Georgia Institute of Technology

Random Access talk, Salishan 2023

BSP Model and Increasing Impact of Idle Time

- The Bulk Synchronous Parallelism model has served us well for decades, but the fraction of idle time is increasing due to an increase in waiting time related to synchronous operations …
  - Waiting for memory operations
  - Waiting for communications
  - Waiting at a barrier
  - Waiting for accelerator kernels
  - Waiting for I/O
- … and the impact of waiting time is increasing rapidly with
  - increasing degree of parallelism
  - increasing variability and load imbalance due to heterogeneity, sparsity, virtualization, …

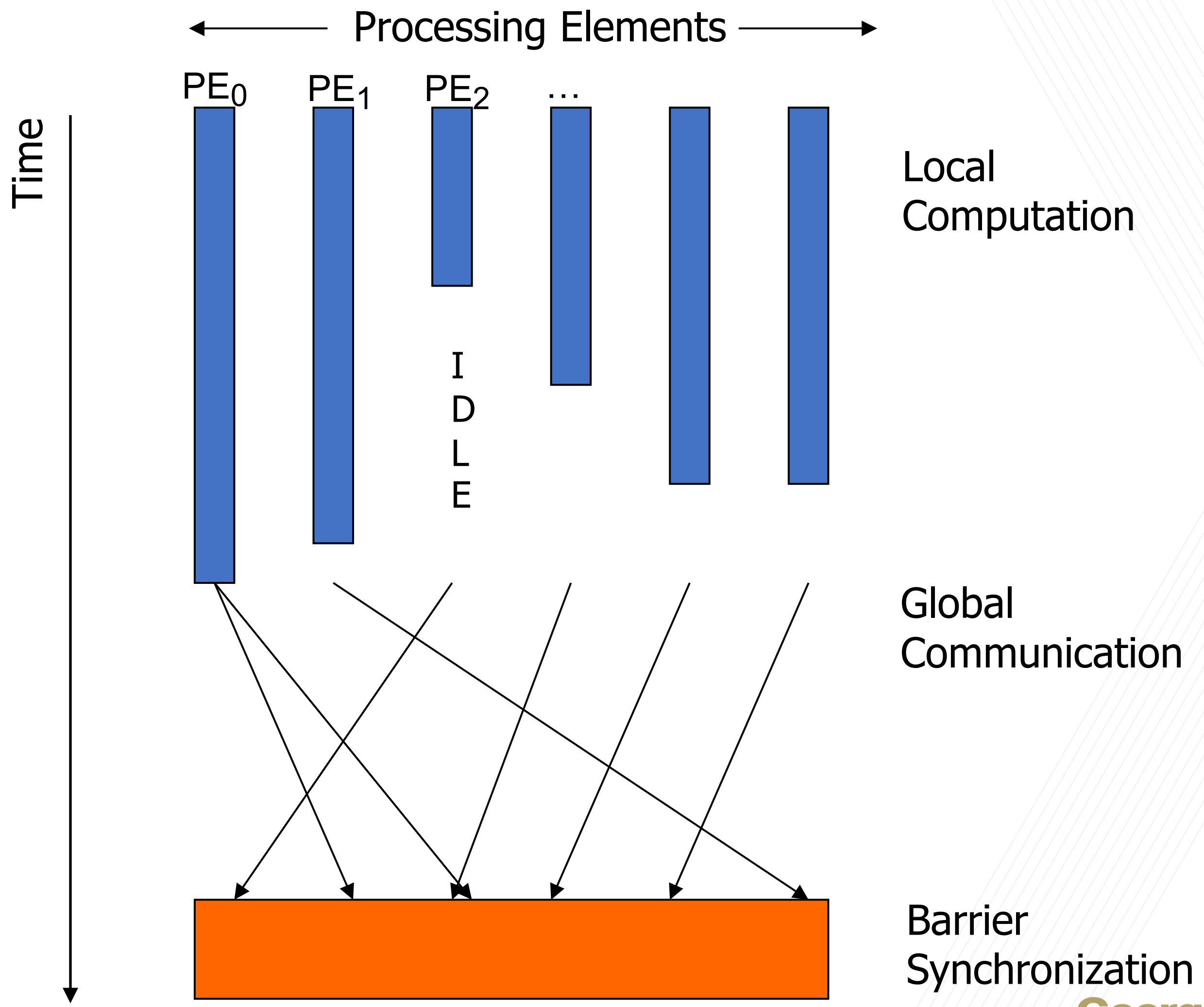


Figure acknowledgment: "An Overview of the BSP Model of Parallel Computation", Michael C. Scherger, Kent State University

Using HPMs to measure idle cycles …

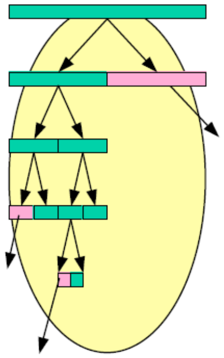
BALE KERNEL	CYCLES	PAPI_RES_STL	% IDLE CYCLES
histo_agp (synchronous)	2.74E+10	1.25E+10	45.6%
histo_selector (asynchronous)	2.16E+09	1.48E+08	6.9%
ig_agp (synchronous)	2.30E+11	4.24E+10	18.4%
ig_selector (asynchronous)	3.52E+10	4.22E+08	1.2%

Preparing for an Asynchronous Future in HPC

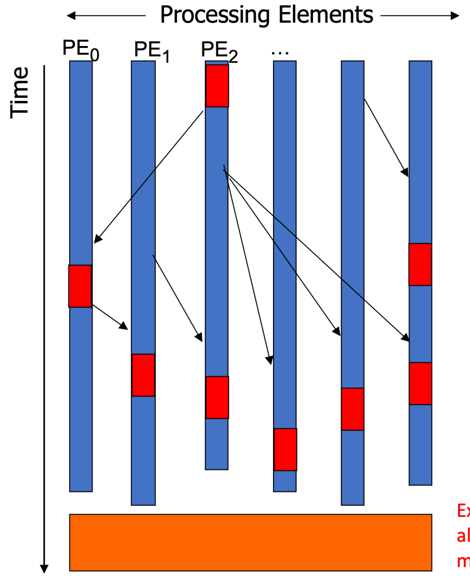
- Ideas from X10 project in HPCS program and follow-on Habanero project at Rice and Georgia Tech
  - async <stmt> creates an asynchronous computation/accelerator/communication task
  - finish <stmt> waits for all tasks in finish scope
- Extend to remote asynchronous tasks
  - async at(<place>>) <stmt>
  - send(<place>, <stmt>)
    - Like an actor/selector model for HPC
- Relax barriers to point-to-point synchronization
  - Dataflow, DAG parallelism, event-driven tasks
  - Doacross
  - Futures/Promises
  - Phasers
- Move towards a Fine-grained-Asynchronous Bulk-Synchronous Parallelism (FA-BSP) model

- "A Fine-grained Asynchronous Bulk Synchronous parallelism model for PGAS applications", JCS 2023.

```
void refine(final int n, final int l, final int nmax) {
        left = new Tree(this,2.0*l);
        right =new Tree(this, 2.0*l+1);
        final nullable Tree ll = left, rr=right;
        if (n < (nmax-1)) {
                async {ll.refine(n+1,2*l,nmax);}
                async { rr.refine(n+1,2*l+1,nmax);}
        }
        if (n < nmax) data = null;
}
```



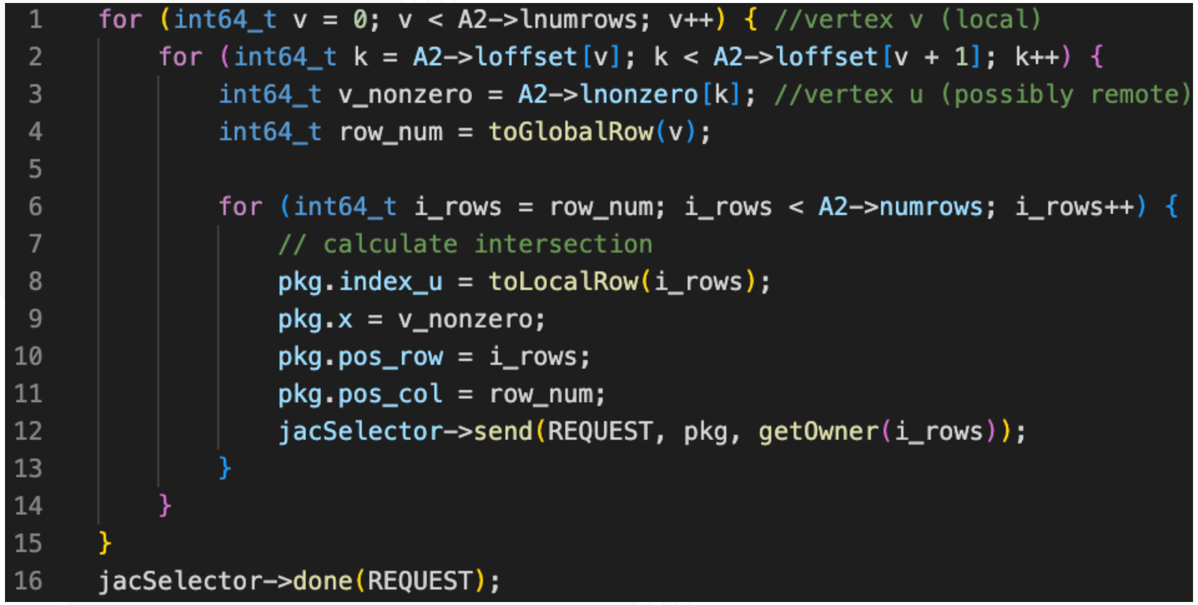
From "What's in it for the Users? Looking Toward the HPCS Languages and Beyond", D. Bernholdt, W.R. Elwasif, Robert J. Harrison, PGAS 2006



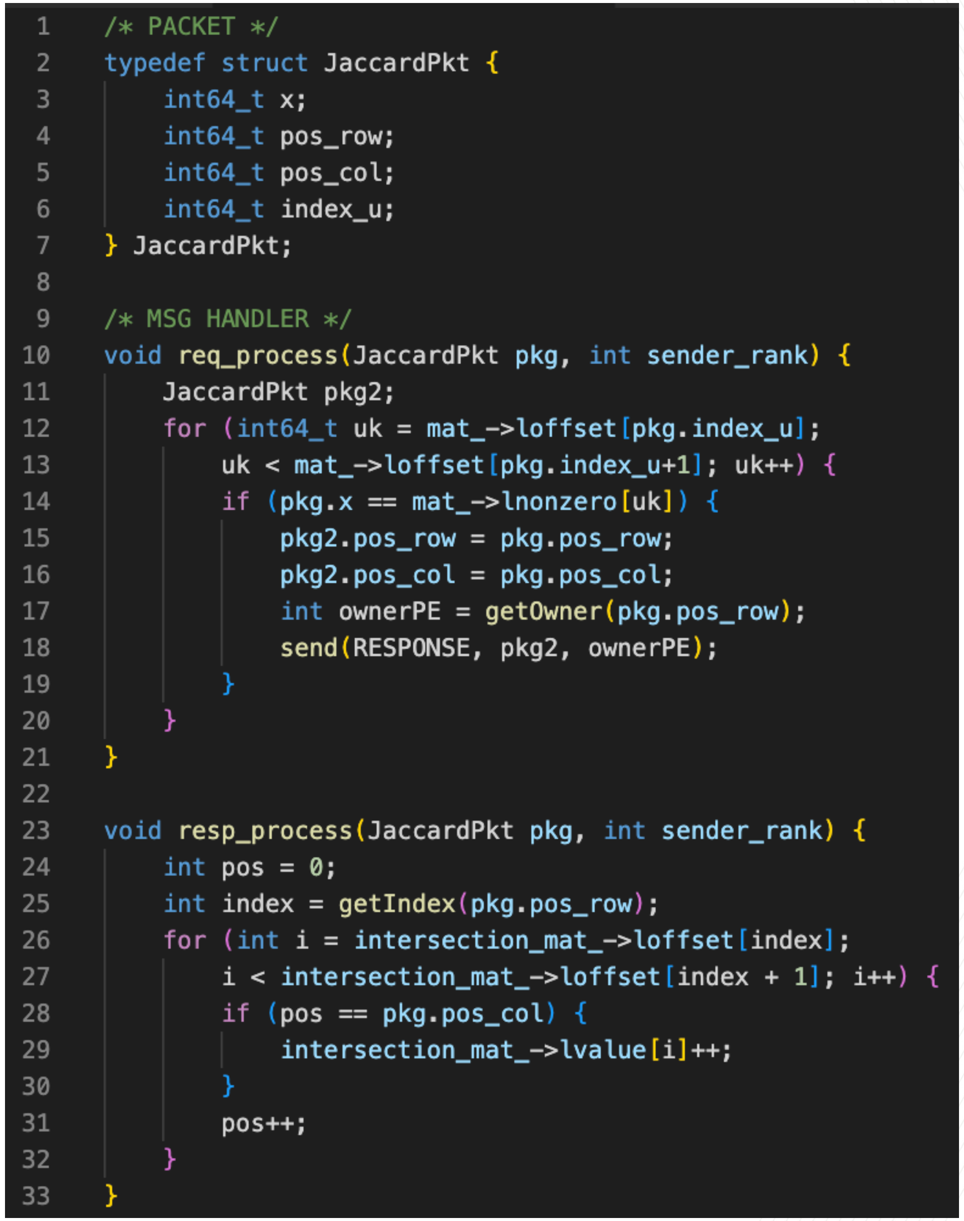
Local Computation + Offloaded Active Messages

Extended Barrier (waits for all processes & all active messages to complete)

Jaccard Benchmark using Actors/Selectors



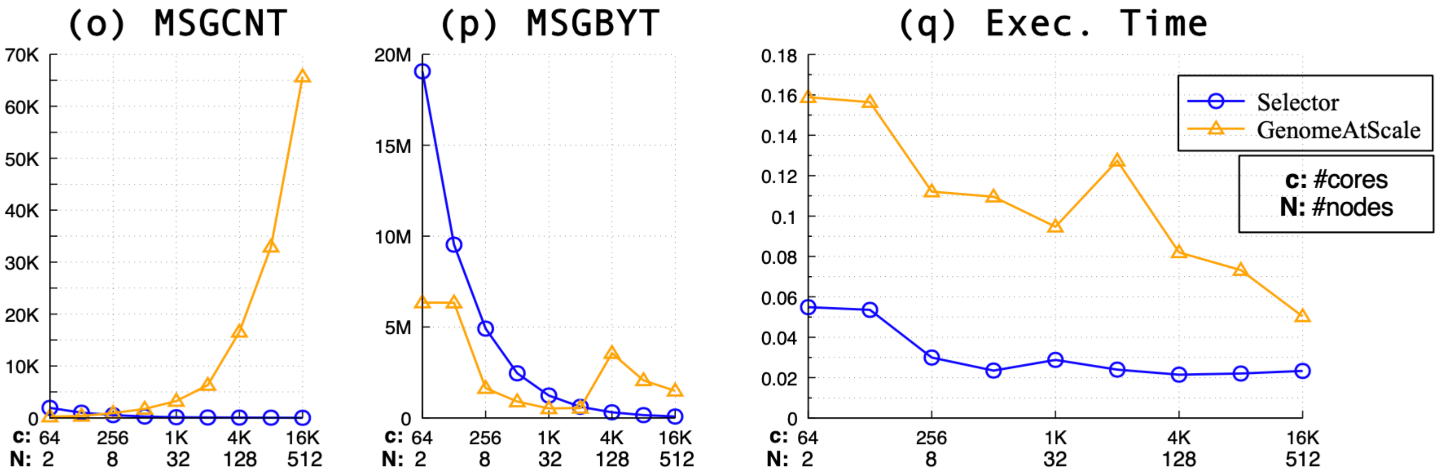
```
for (int64_t v = 0; v < A2->lnumrows; v++) { //vertex v (local)
    for (int64_t k = A2->loffset[v]; k < A2->loffset[v + 1]; k++) {
        int64_t v_nonzero = A2->lnonzero[k]; //vertex u (possibly remote)
        int64_t row_num = toGlobalRow(v);

        for (int64_t i_rows = row_num; i_rows < A2->numrows; i_rows++) {
            // calculate intersection
            pkg.index_u = toLocalRow(i_rows);
            pkg.x = v_nonzero;
            pkg.pos_row = i_rows;
            pkg.pos_col = row_num;
            jacSelector->send(REQUEST, pkg, getOwner(i_rows));
        }
    }
}
jacSelector->done(REQUEST);
```

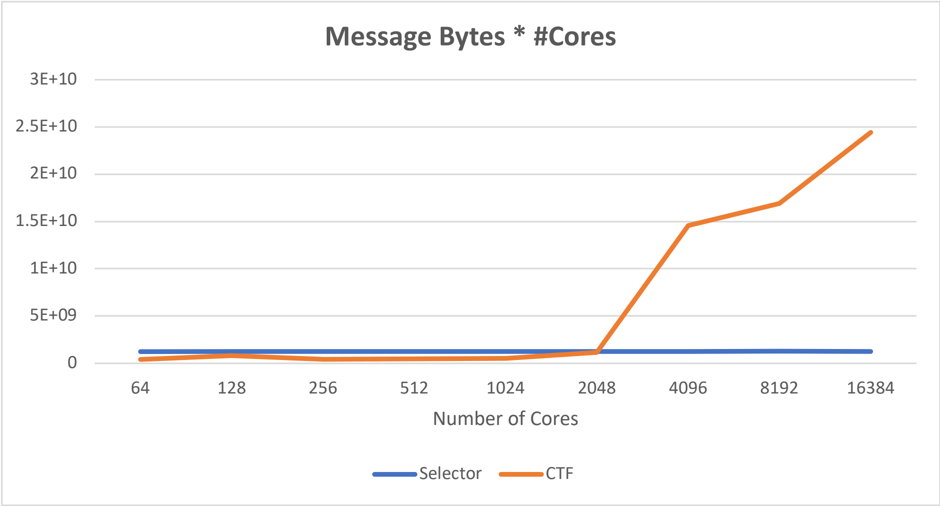


```
/* PACKET */
typedef struct JaccardPkt {
    int64_t x;
    int64_t pos_row;
    int64_t pos_col;
    int64_t index_u;
} JaccardPkt;

/* MSG HANDLER */
void req_process(JaccardPkt pkg, int sender_rank) {
    JaccardPkt pkg2;
    for (int64_t uk = mat_->loffset[pkg.index_u];
        uk < mat_->loffset[pkg.index_u+1]; uk++) {
        if (pkg.x == mat_->lnonzero[uk]) {
            pkg2.pos_row = pkg.pos_row;
            pkg2.pos_col = pkg.pos_col;
            int ownerPE = getOwner(pkg.pos_row);
            send(RESPONSE, pkg2, ownerPE);
        }
    }
}

void resp_process(JaccardPkt pkg, int sender_rank) {
    int pos = 0;
    int index = getIndex(pkg.pos_row);
    for (int i = intersection_mat_->loffset[index];
        i < intersection_mat_->loffset[index + 1]; i++) {
        if (pos == pkg.pos_col) {
            intersection_mat_->lvalue[i]++;
        }
        pos++;
    }
}
```

Preliminary Strong Scaling Results for Jaccard on Perlmutter

- Selector version is faster than Sparse Jaccard-CTF (SCALE=14, Strong Scaling)
- Performed in-depth performance comparisons

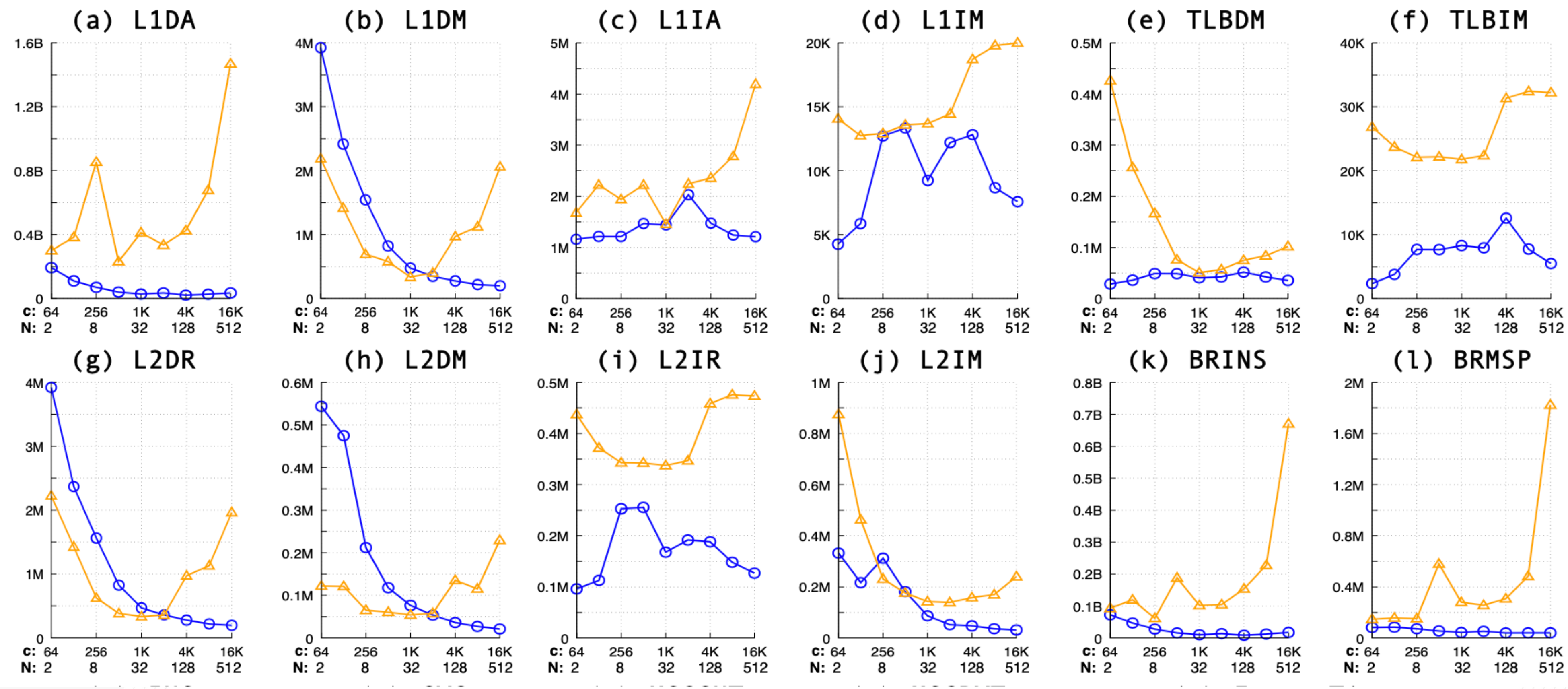




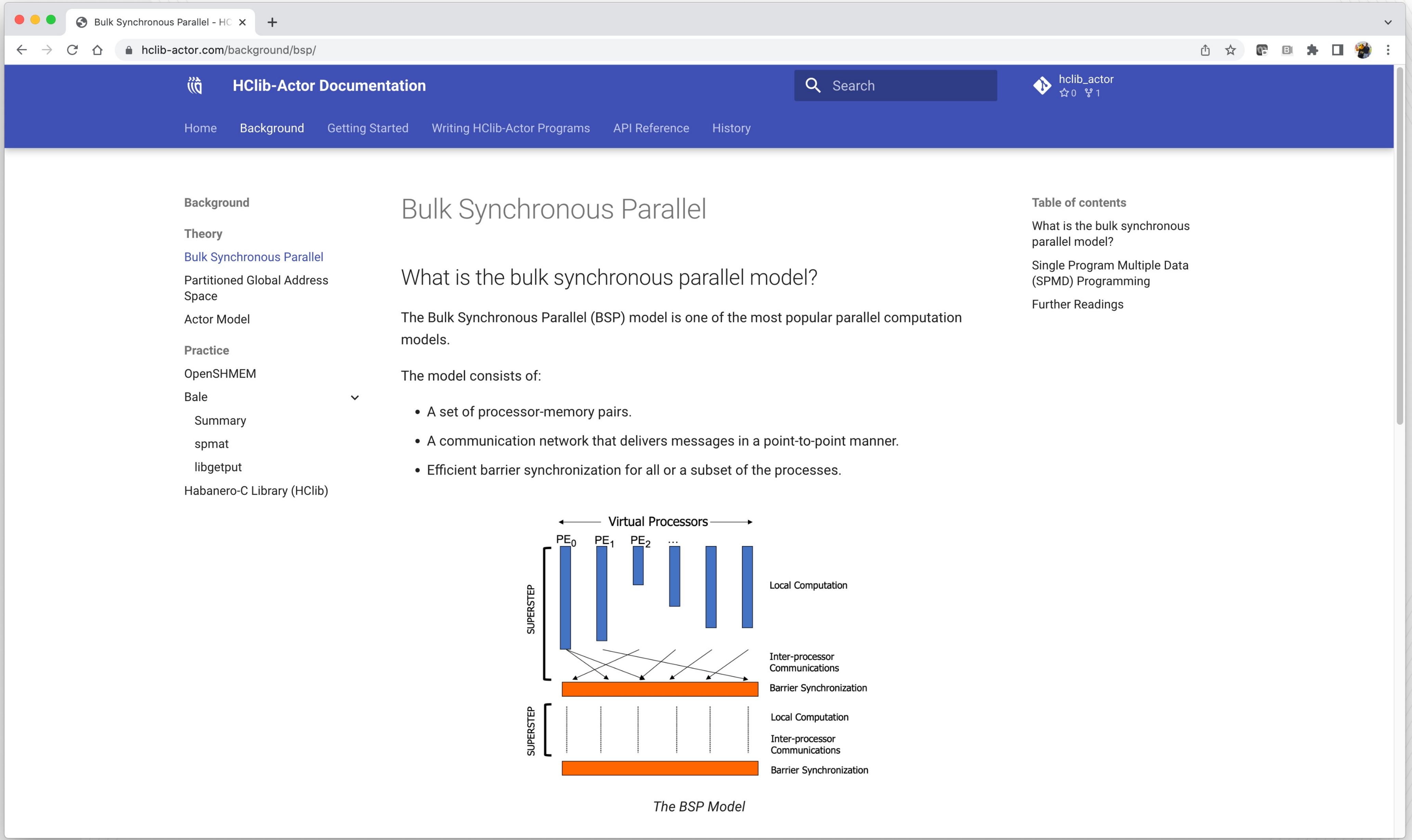
- Key takeaways
  - The CTF version suffered from significant load imbalance mainly in MPI all-to-all ops
  - MSGCNT and MSGBYT decrease as # of PEs increases in Selector. This is not case with CTF version in part because it occasionally performs all-to-all ops for redistribution.
  - We also saw significant decrease of other HPWCs in our version (next slide)

Preliminary Results (contd)

- Other HWPC counter numbers



You can try this at home … just visit hclib-actor.com !



Bulk Synchronous Parallel - HC
hclib-actor.com/background/bsp/
HClib-Actor Documentation
Search
hclib_actor
☆0 1
Home
Background
Getting Started
Writing HClib-Actor Programs
API Reference
History

Background

Theory

- Bulk Synchronous Parallel
- Partitioned Global Address Space
- Actor Model

Practice

- OpenSHMEM
- Bale
  - Summary
  - spmat
  - libgetput
- Habanero-C Library (HClib)

Bulk Synchronous Parallel

What is the bulk synchronous parallel model?

The Bulk Synchronous Parallel (BSP) model is one of the most popular parallel computation models.

The model consists of:

- A set of processor-memory pairs.
- A communication network that delivers messages in a point-to-point manner.
- Efficient barrier synchronization for all or a subset of the processes.

The BSP Model

Conclusion: Prepare for an Asynchronous Future!

- Replace synchronous algorithms by asynchronous algorithms
- Replace task sequencing by asynchronous tasks with task dependences
- Replace blocking accelerator kernel offloads by asynchronous offloads
- Replace blocking communications by asynchronous/nonblocking communications, including actor messages
- Replace barriers by point-to-point synchronization
  - DAG parallelism, Dataflow, Event-driven tasks, Doacross, Futures/Promises, Phasers
- This trend can also be seen at the OS level (e.g., io_uring asynchronous I/O API for Linux) and is motivating a fresh look at the hardware level (e.g., asynchronous circuits bridging heterogeneous processors)
- The move towards an asynchronous future for HPC is well under way!