Codesign Proxy Apps in UPC

Steven Hofmeyr

Lawrence Berkeley National Laboratory

June 2013
DEGAS Summer Retreat

BERKELEY LAB

Which Apps?

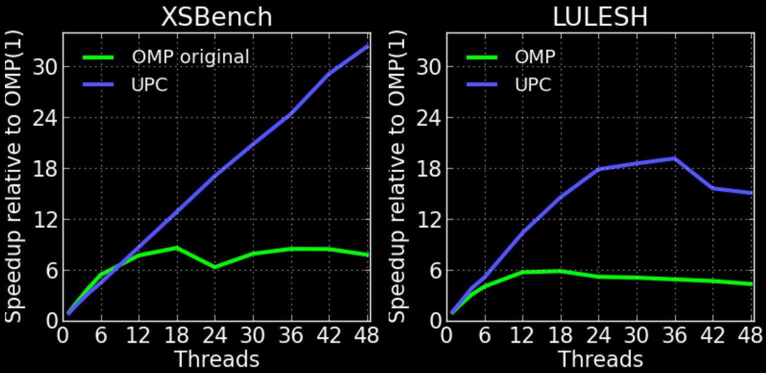
- Small code base (incl. libs)
- C/C++ with OpenMP
- Different co-design centers

Chosen:

- CESAR
  - XSBench v3
  - 653 LOC
- ExMatEx
  - LULESH v1.0.1
  - 2350 LOC
- Exact
  - MG (others)

App Scaling

Original OMP vs best tuned UPC:



Test platform: 48-core AMD Opteron 6174, 8 NUMA nodes, 128GB Mem

Converting OpenMP to UPC

Some parts are straightforward:

- **`#pragma omp parallel for`**

  -> **`upc_forall`**
- **`#pragma omp critical`**

  -> **`bupc_allv_reduce_all()`**

Memory locality is not:

- When should memory be shared? (**`shared`**)
- When should memory be blocked? (**`shared []`**)

Memory conversion strategy:

- private whenever possible
- replicate to prevent sharing

BERKELEY LAB

XSBench

- Monte Carlo simulation of paths of neutrons traveling across a reactor core

  –> 85% of runtime in calculation of macroscopic neutron cross sections

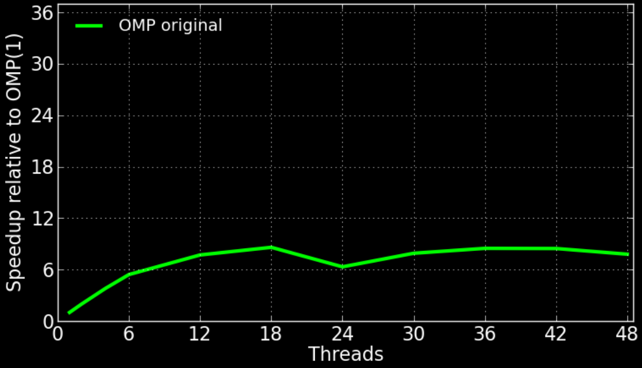
```
random_sample
binary_search
for each nuclide
    lookup_bounding_micro_xs
    interpolate
    accumulate_macro_xs
```

- Embarrassingly parallel
- But uses lots of memory

XSBench OMP Doesn't Scale

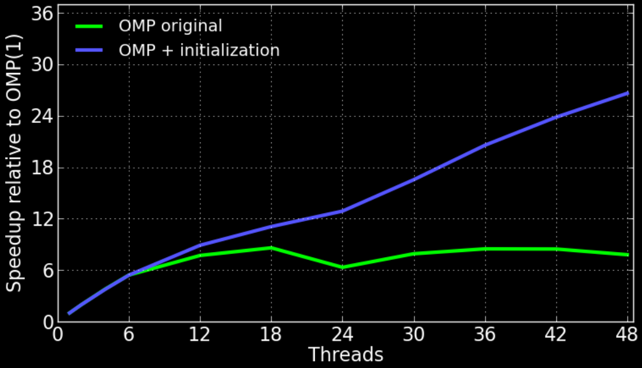
- Option to add flops; according to the README:

"Adding flops has so far shown to increase scaling, indicating that there is in fact a bottleneck being caused by the memory loads."



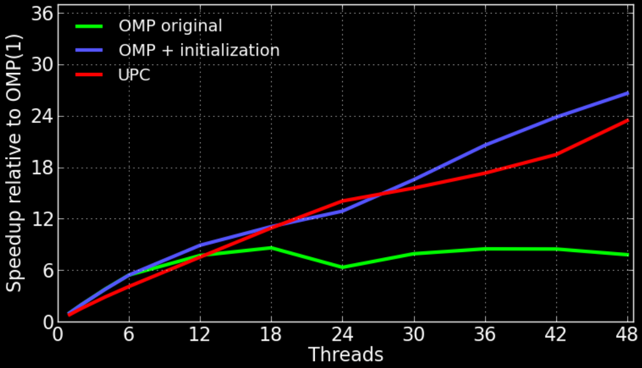
XSBench OMP Initialization

- But memory locality is the problem (on NUMA)
- Adding parallel initialization makes it scale



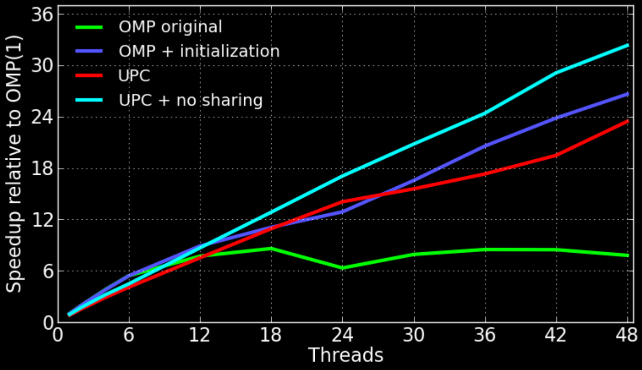
XSBench UPC

- Private replication of data
- Except: make largest memory array shared



XSBench UPC No Shared Mem

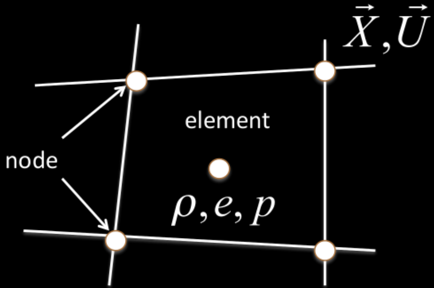
- Improves if we make all memory private
- Doesn't scale to large problem sizes
    - 355 isotopes requires 60GB for full repl. on 48 cores



BERKELEY LAB

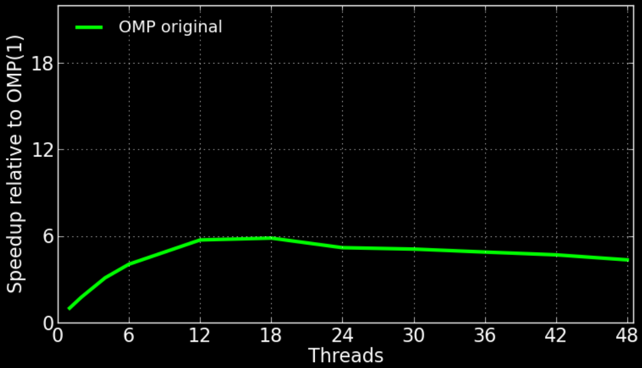
LULESH

- Models explicit hydrodynamics portion of ALE3D
- Particular application is a Sedov blast wave problem
- Used to explore various programming models, e.g. Charm++, Chapel, Loci, Liszt
- Solves equations on a staggered 3D spatial mesh
- Most communication is nearest neighbor on a hexahedral 3D grid



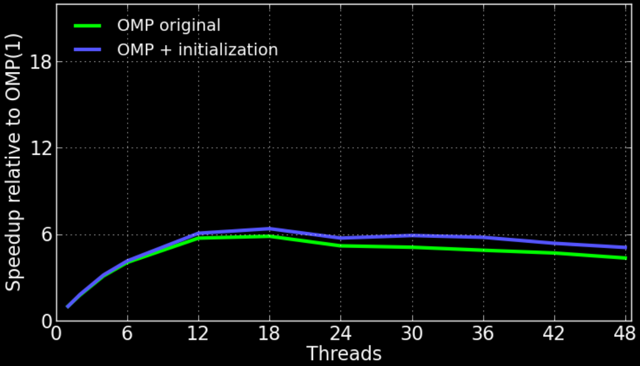
LULESH OMP

- Doesn't scale beyond 12 cores (2 NUMA nodes)



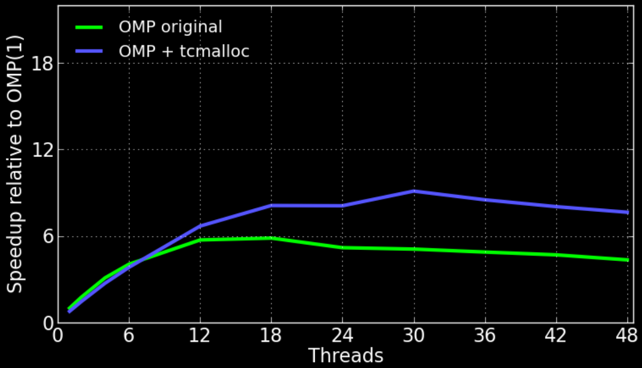
LULESH OMP Parallel Initialization

- Parallel initialization helps only slightly
- Still doesn't scale beyond 18 cores
- Uses temporary arrays with **`malloc`** and **`free`** in many calls



LULESH OMP TCMalloc

- Liu et al (Rice) improve performance with TCMalloc:
  - **`free`** in glibc releases pages to OS
  - subsequent calls to **`malloc`** → OS zero-fills pages
  - TCMalloc doesn't return **`free`**'d pages to the OS
- TCMalloc slow for < 6 threads (e.g. 1 core 1.28x)



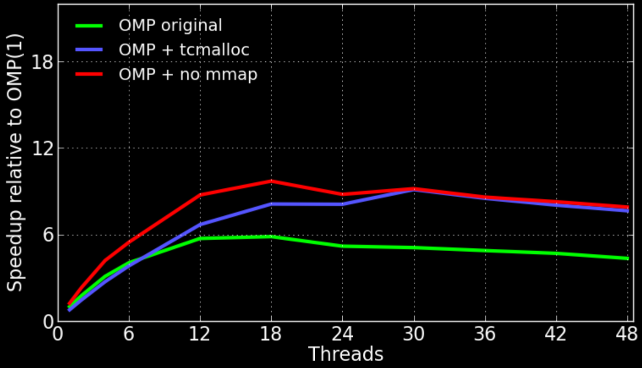
BERKELEY LAB

LULESH OMP Avoid **`mmap`**

- For larger mem, glibc uses **`mmap`** instead of **`brk`**
- Force glibc to always use **`brk`** with flags:

`MALLOC_MMAP_MAX_=0`

`MALLOC_TRIM_THRESHOLD_=-1`



LULESH UPC

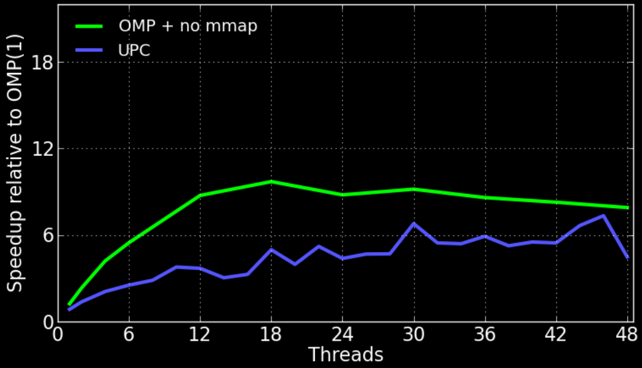
- LULESH authors advise:

  "Do not make simplifications"
- None-the-less, I made some simplifications:
  - Primarily for readability and clarity
  - Why follow certain implementation choices? (e.g. using temp arrays)
- Performance improvements in UPC at scale
  - primarily due to locality management, not simplifications
- UPC with one thread is slower than C++ serial
  - Best UPC 298s, best C++ serial 283s

BERKELEY LAB

LULESH Naive UPC

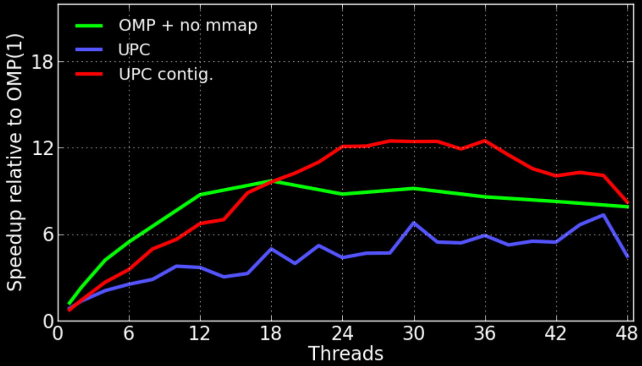
- Replicate data to make it private where possible
- Shared arrays distributed cyclically (default)
- Poor compared to OMP



LULESH UPC Memory Layout

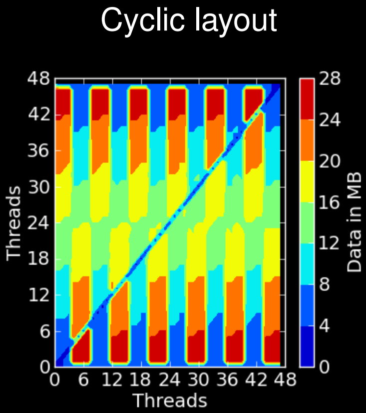
- Cyclic layout poor fit for communication pattern
- Contiguous layout (blocked) reduces communication

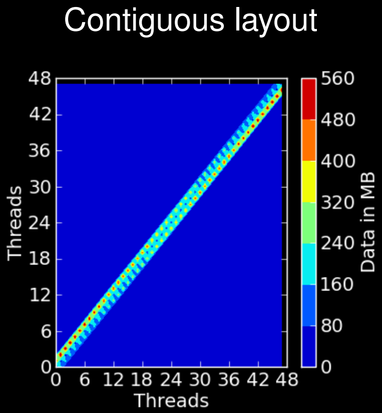
```
shared [*] double x[N * THREADS];
```



BERKELEY LAB

LULESH UPC Communication

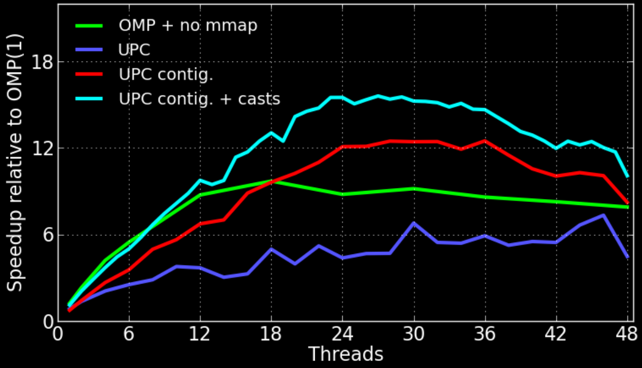




LULESH UPC Cast Shared to Private

- Use private pointer to the thread block in shared array

```
double* my_x = (double*)(x + MYTHREAD * BSIZE)
```

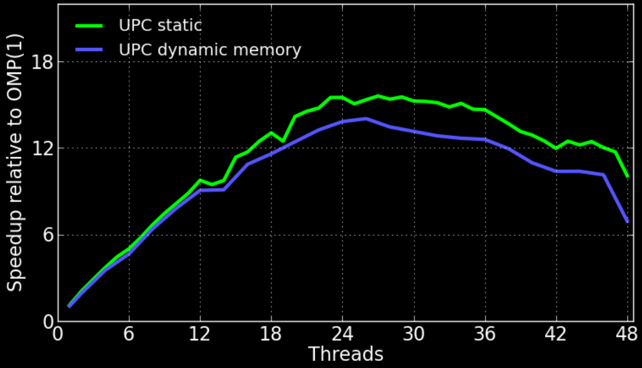


LULESH UPC Dynamic vs Static Mem

- Dynamic memory allocation worse than static
- From upcc man page, static has:

"potential for more aggressive compiler optimizations"

- But 48 is not a power-of-two



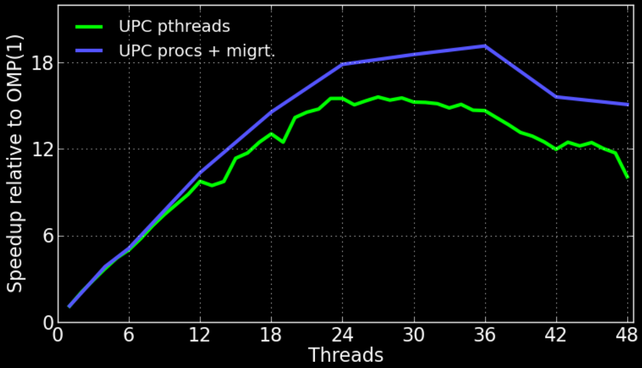
BERKELEY LAB

LULESH UPC Procs vs Pthreads

- One thread per process (one per core) is faster
- With procs, can pin threads and migrate static pages

```
migrate_pages(pid, maxnode, oldnodes, newnodes)
```

- (But only migrates private pages)



LULESH UPC Procs vs Pthreads

- At 48 cores, pthreads takes 33s, processes takes 22s.
- Top non-app code functions with pthreads:

  `upcr_wait_internal` 15%
  `__ticket_spin_lock` 3% (kernel)
  `gasnete_coll_broadcast` 2%
  `gasnete_coll_gather` 2%

- Top non-app code functions with pinned procs:

  `gasneti_AMPSHMPoll` 5%
  `gasnete_pshmbarrier_wait` 5%

- For comparison, collectives with pinned procs:

  `gasnete_coll_broadcast` 0.2% (15x)
  `gasnete_coll_gather` 0.04% (75x)

BERKELEY LAB

Lessons Learned

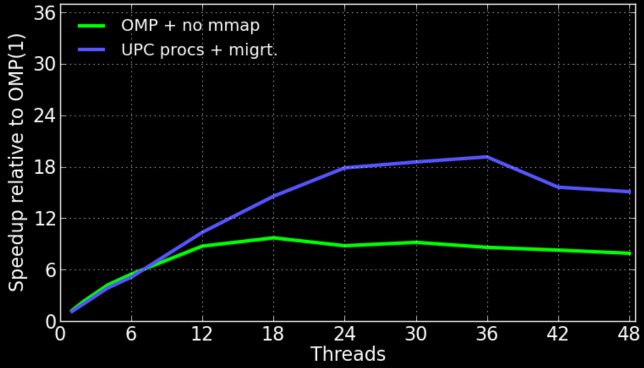
- On a large NUMA system, managing remote memory access is key

  –> good preparation for distributed memory?
- Parallel initialization in OMP for locality
- UPC:
  - Replication to private can help, but limited by available memory $\rightarrow$ replicate fixed amount?
  - Explicitly cast to private whenever possible
  - Contiguous blocking can be effective at reducing communication
  - With static memory, need to migrate pages after pinning
  - Procs can be significantly better than pthreads

BERKELEY LAB

A Final Mystery

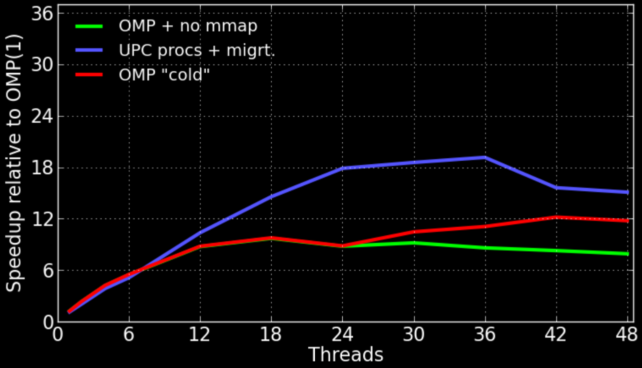
LULESH best scaling on 48 cores:



BERKELEY LAB

OMP Scales Better "Cold"

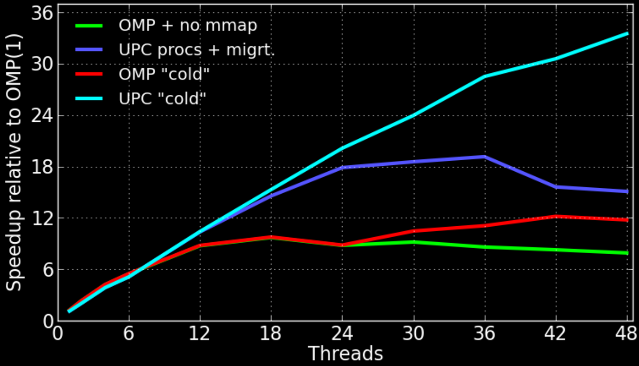
After system restarts, OMP scales better



UPC Scales a Lot Better "Cold"

And an even more dramatic improvement for UPC

- For a while anyway
- After several runs, reverts to slow "hot" performance: 10s → 13s → 15s → 18s → 22s



BERKELEY LAB

NUMA?

- Find out what pages are mapped to what nodes for a process from **`/proc/self/numa_maps`**
- No difference between local and remote mappings for hot and cold
  - Shared pages map 0.37 local, 0.37 near, 0.26 far
  - Private pages map 1.0 local, 0.0 near, 0.0 far
- But NUMA seems to matter still
- Restrict memory to nodes 0-3:
  - runtime cold 10s $\rightarrow$ 25s
  - runtime hot 22s $\rightarrow$ 30s
- Don't see it in XSBench
- Killing UPC accelerates future transitions to hot
- Time spent in kernel $< 2\%$

BERKELEY LAB

Perf Counter Comparisons

Counter	"Cold"	"Hot"
Instructions/cycle	1.20	0.71
Stalled Instructions/cycle	0.41	0.99
Stalled cycles, frontend	3.8E+10	3.7E+10
Stalled cycles, backend	6.4E+11	1.7E+12
LS Buffer dispatch stalls	4.1E+9	2.0E+10
Cache refs	3.1E+11	3.4E+11
Cache misses	1.2E+8	1.3E+8
Ctx switches	3979	5289
Minor faults	3.3E+5	3.3E+5
TLB misses	3.1E+11	3.3E+11
node loads	4.3E+10	4.3E+10
node misses	5.0E+9	5.3E+9

BERKELEY LAB

Questions?

Ideas?

All Codesign Proxy Apps

Name	Languages	LOC
ExMatEx		
CoMD	C++/OCL	2548
HILO 1D/2D	C/MPI	5003
LULESH	C++/OMP	2350
VPFFT	C++/OMP	2637
Exact		
CNS_NoSpec	F90/OMP/MPI	787
MultiGrid_C	C++/OMP/MPI	1704
Cesar		
mocfe_bone	F90/MPI	6252
nekbone	F90/MPI	30105
XSBench	C/OMP	663

Codesign Proxy App Details

Name	Time (s)	Mflps	Mips	VM	RSS	Spd
ExMatEx						
CoMD	192	229	2979	37	12	35.9
HILO 2D	50	563	1900	41	3	37.3
LULESH	342	1079	2303	121	89	4.4
VPFFT	70	622	2387	72	36	3.8
Exact						
CNS_NoSpec	35	795	1994	599	553	13.6
MultiGrid_C	90	577	2257	2553	2474	21.5
Cesar						
mocfe_bone	132	1096	3069	2366	2323	15.3
nekbone	244	1460	3157	927	272	27.0
XSBench	49	224	1798	287	260	7.8

Test platform: 48-core AMD Opteron 6174, 8 NUMA nodes, 128GB Mem