Evaluating the Efficiency of OpenMP Tasking for Unbalanced Computation on Diverse CPU Architectures

PRESENTED BY

Stephen Olivier

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

OpenMP Tasks Still Seeing Limited Adoption

Task construct first added to OpenMP spec. in version 3.0 (2008)

- Continued feature development in subsequent versions of OpenMP
- Tasking model now widely used in the context of asynchronous offload to devices

Slow adoption of tasking in other scenarios – why?

- Concerns about overhead costs of task creation, scheduling, synchronization
- Users unsure about appropriate granularity of tasks to use
- Perceived variation in quality of implementations across vendors and platforms

Contributions of this paper/presentation:

- Benchmark a challenging task parallel computation
- …on four different architectures
- …using LLVM/Clang and commercial OpenMP implementations
- …to address these questions

Unbalanced Tree Search (UTS) as a Tasking Stress Test

UTS benchmark for dynamic load balancing of computations

- First proposed and demonstrated in LCPC 2006 paper
- OpenMP tasking and Cilk applied to UTS in IWOMP 2009 paper
- Compared to Cilk++ (Cilk Plus forerunner) and TBB in follow-up IJPP article
- Added to Barcelona OpenMP Tasks Suite (BOTS)

UTS problem: count nodes of a dynamically-generated tree

- Tree implicitly generated on-the-fly by sampling a binomial probability distribution
- Each non-root tree node has ***m*** children with probability ***q***, none with probability *1-q*
- Do ***g*** repeats of SHA-1 hash at each tree node (the work)

Resulting computation is unpredictably imbalanced

- Size of subtree rooted at each node not dictated by proximity to root
- Requires continuous dynamic load balancing throughout execution

UTS OpenMP Tasking Implementation (Adapted From BOTS)

```
unsigned long long search(Node *parent, int numChildren)
{
  Node n[numChildren], *nodePtr;
  int i, j;
  unsigned long long subtreesize = 1, partialCount[numChildren];

  // Visit each child
  for (i = 0; i < numChildren; i++) {
     nodePtr = &n[i];

     // The following line is the work (one or more SHA-1 ops)
     for (j = 0; j < granularity; j++)
        sha1_rng(parent->state.state, nodePtr->state.state, i);

     // Sample a binomial distribution to determine the number of children of child i
     nodePtr->numChildren = uts_numChildren(nodePtr);

     if (nodePtr->numChildren > 0)
        // Traverse the subtree rooted at child i to get subtree size
        #pragma omp task untied firstprivate(i, nodePtr) shared(partialCount)
           partialCount[i] = search(nodePtr, nodePtr->numChildren);
     else
        partialCount[i] = 1;    // Leaf node (no new task generated)
  }

  // Wait for all subtree traversals
  #pragma omp taskwait

  // Combine subtree counts from children to get total size of subtree rooted at Node
  for (i = 0; i < numChildren; i++)
    subtreesize += partialCount[i];

  return subtreesize;
}
```

Test Problem Used for Experiments

Parameters

- 2000 children of root node
- Probability of non-root node having children $q = 0.200014$
- Probability of non-root node not having children $(1 - q) = 0.799986$
- Each non-root non-leaf node has 5 children
- Experiments vary number of SHA-1 hash repeats per node

Generated tree

- 111 345 631 total nodes
- 89 076 904 leaf nodes (~80% of the total nodes)
- 22 268 727 non-leaf nodes (~20% of the total nodes)
- Maximum depth of 17 844 nodes

Experimental Setup (Platforms and OpenMP Implementations)

Intel Xeon Skylake (Xeon SKL)

- Dual socket with 24 cores per socket (48 cores total), 2-way SMT
- Compilers: Intel Compiler 19; Clang 9 with LLVM OpenMP runtime
- Also Threading Building Blocks (TBB) with Intel C++ Compiler 19

IBM POWER9 (IBM P9)

- Dual socket with 22 cores per socket (44 cores total), 4-way SMT
- Compilers: PGI Compiler 20.1; Clang 9 with LLVM OpenMP runtime

Arm ThunderX2 (Arm TX2)

- Dual socket with 28 cores per socket (56 cores total), 2-way SMT (enabled)
- Compilers: Arm Compiler 20.0 "armclang"; Clang 9 with LLVM OpenMP runtime

Intel Xeon Phi "Knights Landing" (Xeon Phi)

- Single socket with 68 cores, 4-way SMT
- Compilers: Intel Compiler 19; Cray CCE 9.1.2; Clang 9 with LLVM OpenMP runtime

Varying Task Granularity in UTS

Task granularity dictated by number of SHA-1 hash repeats per tree node

Varied by powers of 2 from 1 to 32 in our experiments

5 children generated per OpenMP task, so 5 to 160 SHA-1 hashes per task

Translations to time and machine instructions shown in tables at right

Table 1. Translating task granularity from SHA-1 operations / task to time / task

Architecture and Implementation	Time (μs) per op.	Time (μs) per recursive call at granularity					
		5 ops.	10 ops.	20 ops.	40 ops.	80 ops.	160 ops.
Xeon SKL - ICC	0.22	1.12	2.23	4.47	8.94	17.9	35.7
Xeon SKL - Clang	0.18	0.89	1.78	3.55	7.10	14.2	28.4
IBM P9 - PGI	0.31	1.53	3.06	6.13	12.2	24.5	49.0
IBM P9 - Clang	0.29	1.45	2.90	5.80	11.6	23.2	46.4
Arm TX2 - Armclang	0.32	1.61	3.22	6.43	12.9	25.7	51.4
Arm TX2 - Clang	0.34	1.73	3.45	6.90	13.8	27.6	55.2
Xeon Phi - ICC	0.64	3.21	6.42	12.8	25.7	51.4	103
Xeon Phi - Clang	0.74	3.68	7.36	14.7	29.4	58.9	118
Xeon Phi - CCE	0.63	3.14	6.29	12.6	25.2	50.3	101

Table 2. Translating task granularity from SHA-1 operations / task to machine instructions / task

Architecture and Implementation	Kilo instr. per op.	Kilo instr. per recursive call at granularity					
		5 ops.	10 ops.	20 ops.	40 ops.	80 ops.	160 ops.
Xeon SKL - ICC	1.74	8.72	17.4	34.9	69.7	139	279
Xeon SKL - Clang	1.70	8.51	17.0	34.0	68.1	136	272
IBM P9 - PGI	1.65	8.26	16.5	33.1	66.1	132	264
IBM P9 - Clang	1.67	8.35	16.7	33.4	66.8	133	267
Arm TX2 - Armclang	1.39	6.97	13.9	27.9	55.7	111	223
Arm TX2 - Clang	1.51	7.59	15.2	30.4	60.7	121	243
Xeon Phi - ICC	1.70	8.51	17.0	34.0	68.1	136	272
Xeon Phi - Clang	1.71	8.57	17.1	34.3	68.6	137	274
Xeon Phi - CCE	1.63	8.15	16.3	32.6	65.2	130	261

Varying Task Granularity in UTS

Task granularity dictated by number of SHA-1 hash repeats per tree node

Varied by powers of 2 from 1 to 32 in our experiments

5 children generated per OpenMP task, so 5 to 160 SHA-1 hashes per task

Translations to time and machine instructions shown in tables at right

Table 1. Translating task granularity from SHA-1 operations / task to time / task

Architecture and Implementation	Time (μs) per op.	Time (μs) per recursive call at granularity					
		5 ops.	10 ops.	20 ops.	40 ops.	80 ops.	160 ops.
Xeon SKL - ICC	0.22	1.12	2.23	4.47	8.94	17.9	35.7
Xeon SKL - Clang	0.18	0.89	1.78	3.55	7.10	14.2	28.4
IBM P9 - PGI	0.31	1.53	3.06	6.13	12.2	24.5	49.0
IBM P9 - Clang	0.29	1.45	2.90	5.80	11.6	23.2	46.4
Arm TX2 - Armclang	0.32	1.61	3.22	6.43	12.9	25.7	51.4
Arm TX2 - Clang	0.34	1.73	3.45	6.90	13.8	27.6	55.2
Xeon Phi - ICC	0.64	3.21	6.42	12.8	25.7	51.4	103
Xeon Phi - Clang	0.74	3.68	7.36	14.7	29.4	58.9	118
Xeon Phi - CCE	0.63	3.14	6.29	12.6	25.2	50.3	101

Wide range

Table 2. Translating task granularity from SHA-1 operations / task to machine instructions / task

Architecture and Implementation	Kilo instr. per op.	Kilo instr. per recursive call at granularity					
		5 ops.	10 ops.	20 ops.	40 ops.	80 ops.	160 ops.
Xeon SKL - ICC	1.74	8.72	17.4	34.9	69.7	139	279
Xeon SKL - Clang	1.70	8.51	17.0	34.0	68.1	136	272
IBM P9 - PGI	1.65	8.26	16.5	33.1	66.1	132	264
IBM P9 - Clang	1.67	8.35	16.7	33.4	66.8	133	267
Arm TX2 - Armclang	1.39	6.97	13.9	27.9	55.7	111	223
Arm TX2 - Clang	1.51	7.59	15.2	30.4	60.7	121	243
Xeon Phi - ICC	1.70	8.51	17.0	34.0	68.1	136	272
Xeon Phi - Clang	1.71	8.57	17.1	34.3	68.6	137	274
Xeon Phi - CCE	1.63	8.15	16.3	32.6	65.2	130	261

Varying Task Granularity in UTS

Task granularity dictated by number of SHA-1 hash repeats per tree node

Varied by powers of 2 from 1 to 32 in our experiments

5 children generated per OpenMP task, so 5 to 160 SHA-1 hashes per task

Translations to time and machine instructions shown in tables at right

Table 1. Translating task granularity from SHA-1 operations / task to time / task

Architecture and Implementation	Time (μs) per op.	Time (μs) per recursive call at granularity					
		5 ops.	10 ops.	20 ops.	40 ops.	80 ops.	160 ops.
Xeon SKL - ICC	0.22	1.12	2.23	4.47	8.94	17.9	35.7
Xeon SKL - Clang	0.18	0.89	1.78	3.55	7.10	14.2	28.4
IBM P9 - PGI	0.31	1.53	3.06	6.13	12.2	24.5	49.0
IBM P9 - Clang	0.29	1.45	2.90	5.80	11.6	23.2	46.4
Arm TX2 - Armclang	0.32	1.61	3.22	6.43	12.9	25.7	51.4
Arm TX2 - Clang	0.34	1.73	3.45	6.90	13.8	27.6	55.2
Xeon Phi - ICC	0.64	3.21	6.42	12.8	25.7	51.4	103
Xeon Phi - Clang	0.74	3.68	7.36	14.7	29.4	58.9	118
Xeon Phi - CCE	0.63	3.14	6.29	12.6	25.2	50.3	101

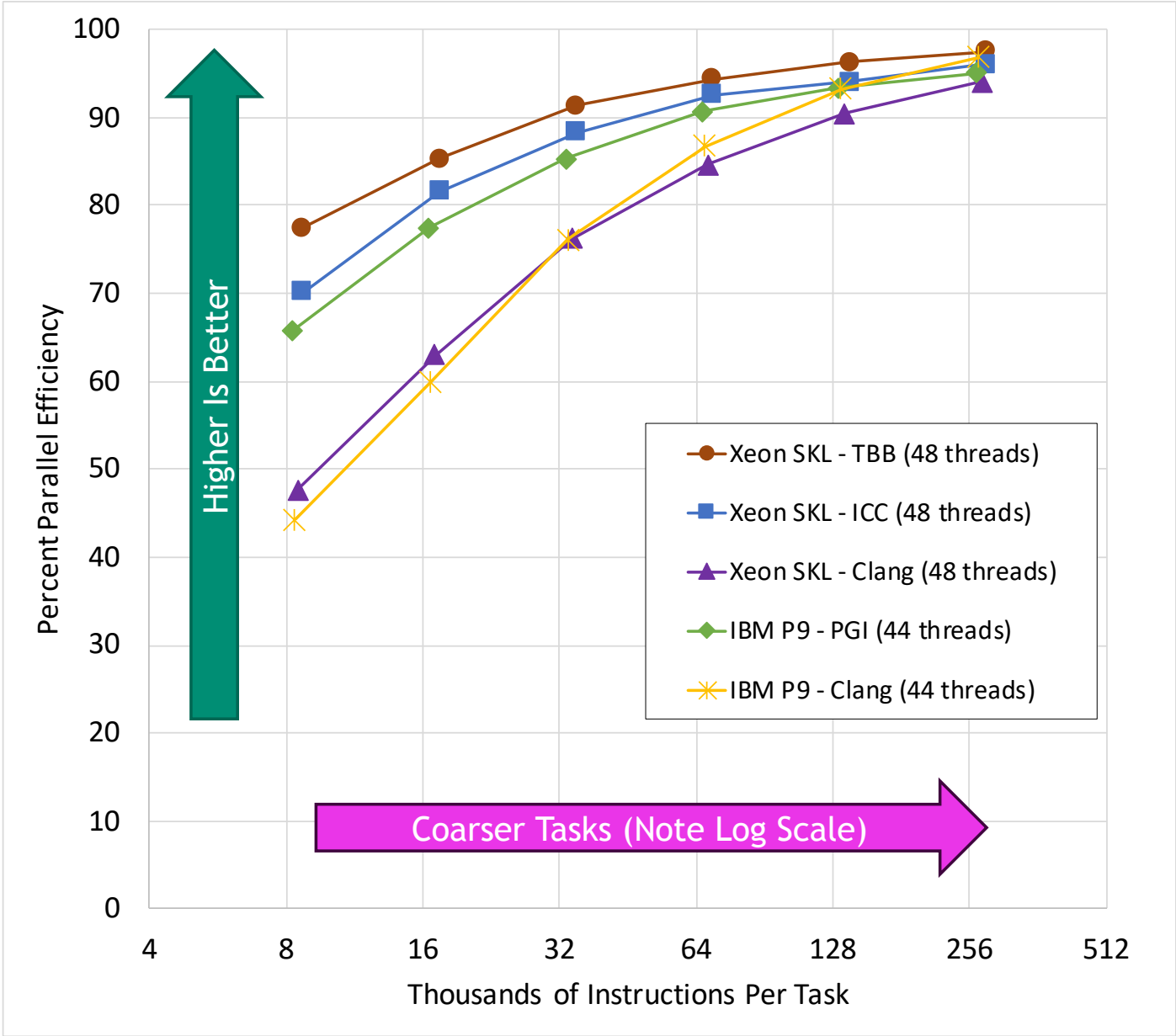
Wide range

Table 2. Translating task granularity from SHA-1 operations / task to machine instructions / task

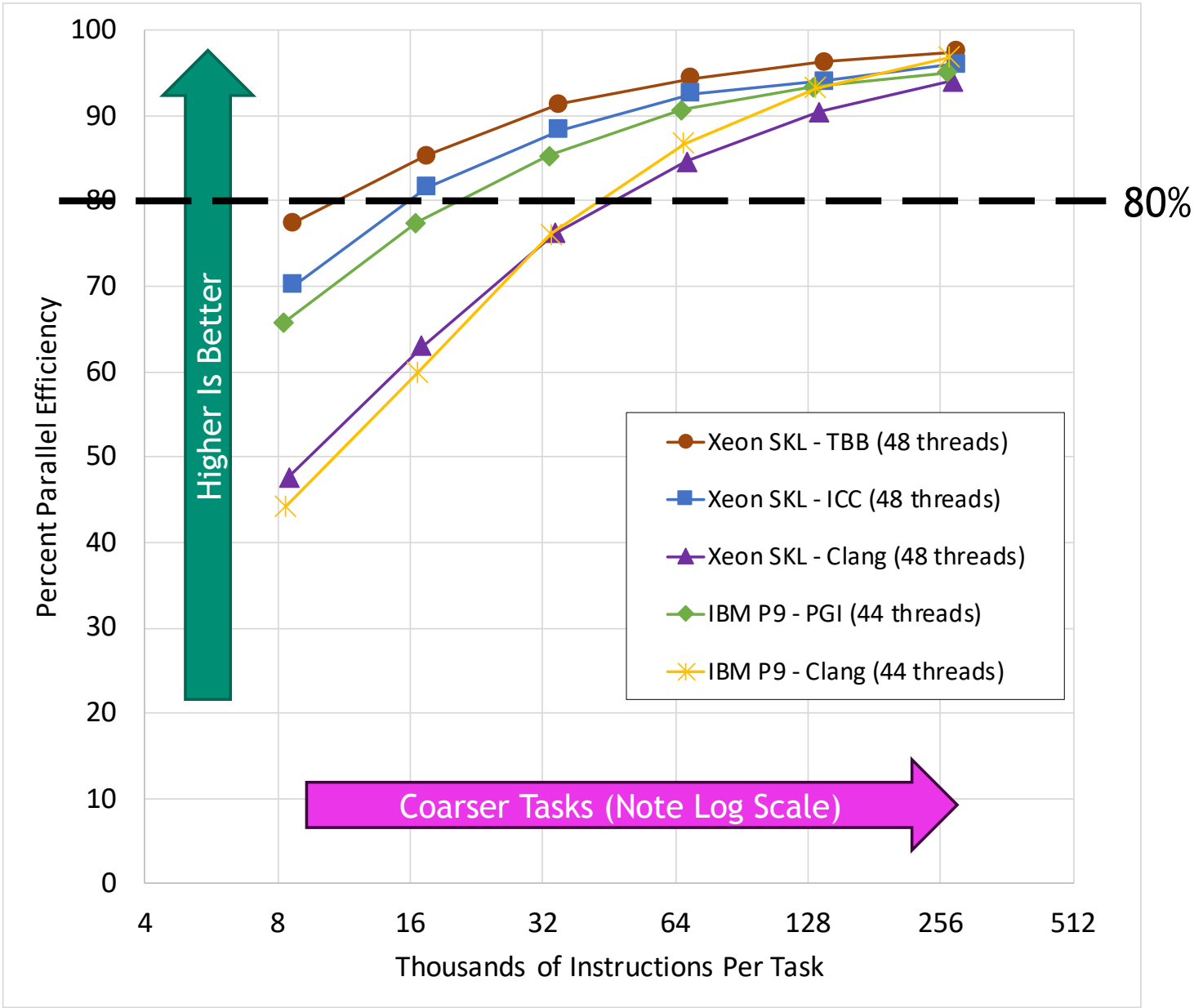
Architecture and Implementation	Kilo instr. per op.	Kilo instr. per recursive call at granularity					
		5 ops.	10 ops.	20 ops.	40 ops.	80 ops.	160 ops.
Xeon SKL - ICC	1.74	8.72	17.4	34.9	69.7	139	279
Xeon SKL - Clang	1.70	8.51	17.0	34.0	68.1	136	272
IBM P9 - PGI	1.65	8.26	16.5	33.1	66.1	132	264
IBM P9 - Clang	1.67	8.35	16.7	33.4	66.8	133	267
Arm TX2 - Armclang	1.39	6.97	13.9	27.9	55.7	111	223
Arm TX2 - Clang	1.51	7.59	15.2	30.4	60.7	121	243
Xeon Phi - ICC	1.70	8.51	17.0	34.0	68.1	136	272
Xeon Phi - Clang	1.71	8.57	17.1	34.3	68.6	137	274
Xeon Phi - CCE	1.63	8.15	16.3	32.6	65.2	130	261

Narrower range

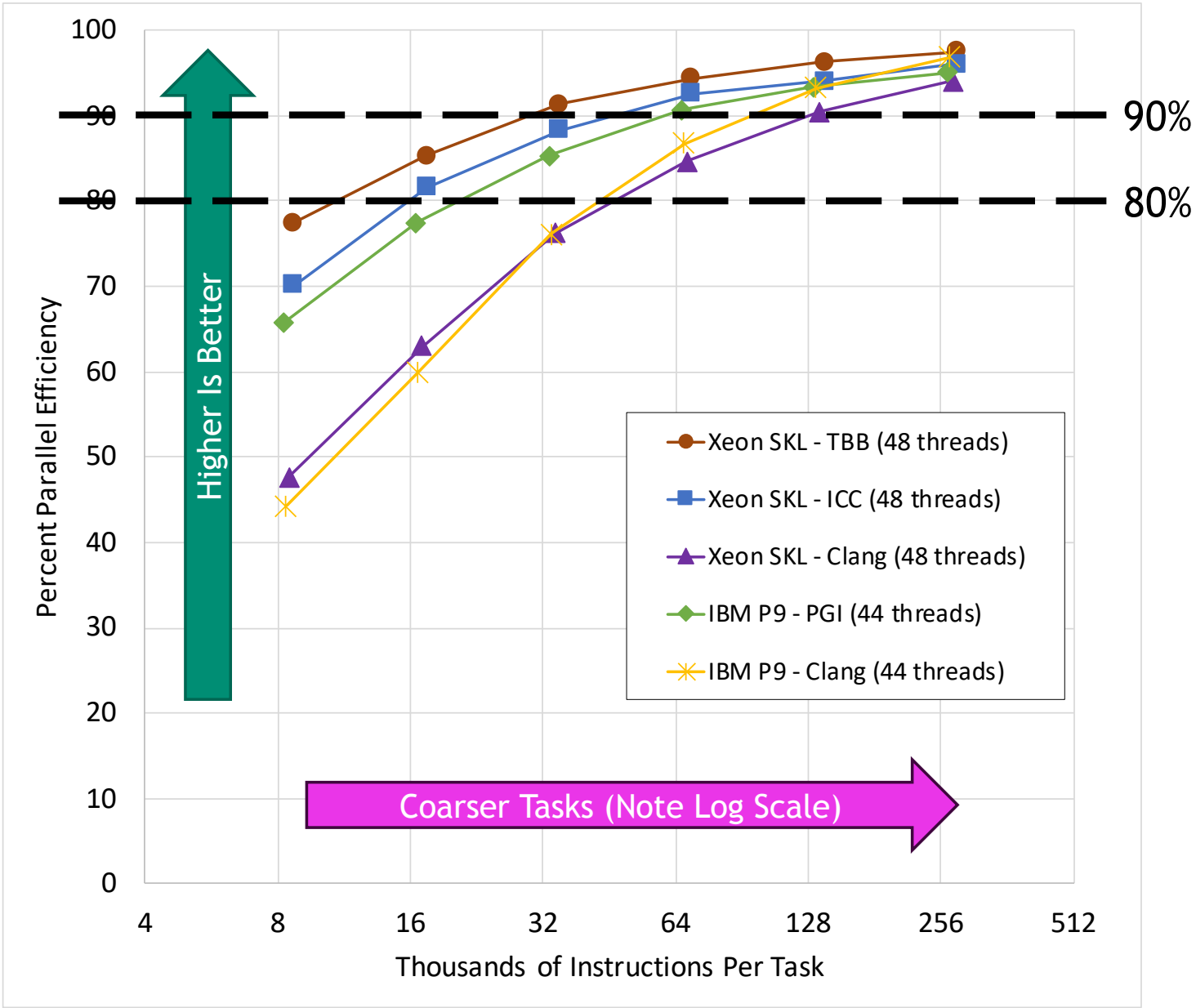
Parallel Efficiency on Intel Xeon & IBM P9 (One Thread/Core)



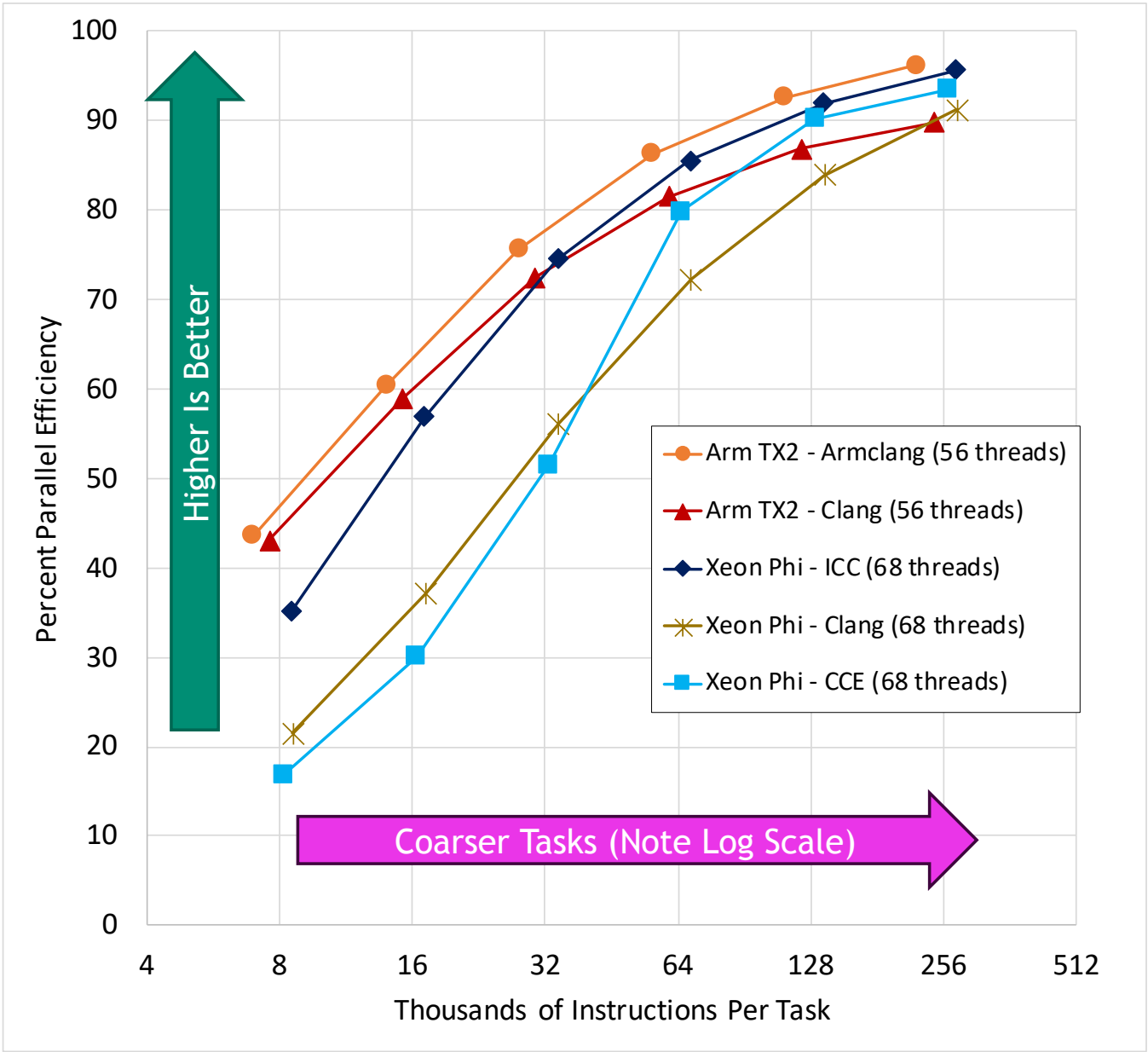
Parallel Efficiency on Intel Xeon & IBM P9 (One Thread/Core)



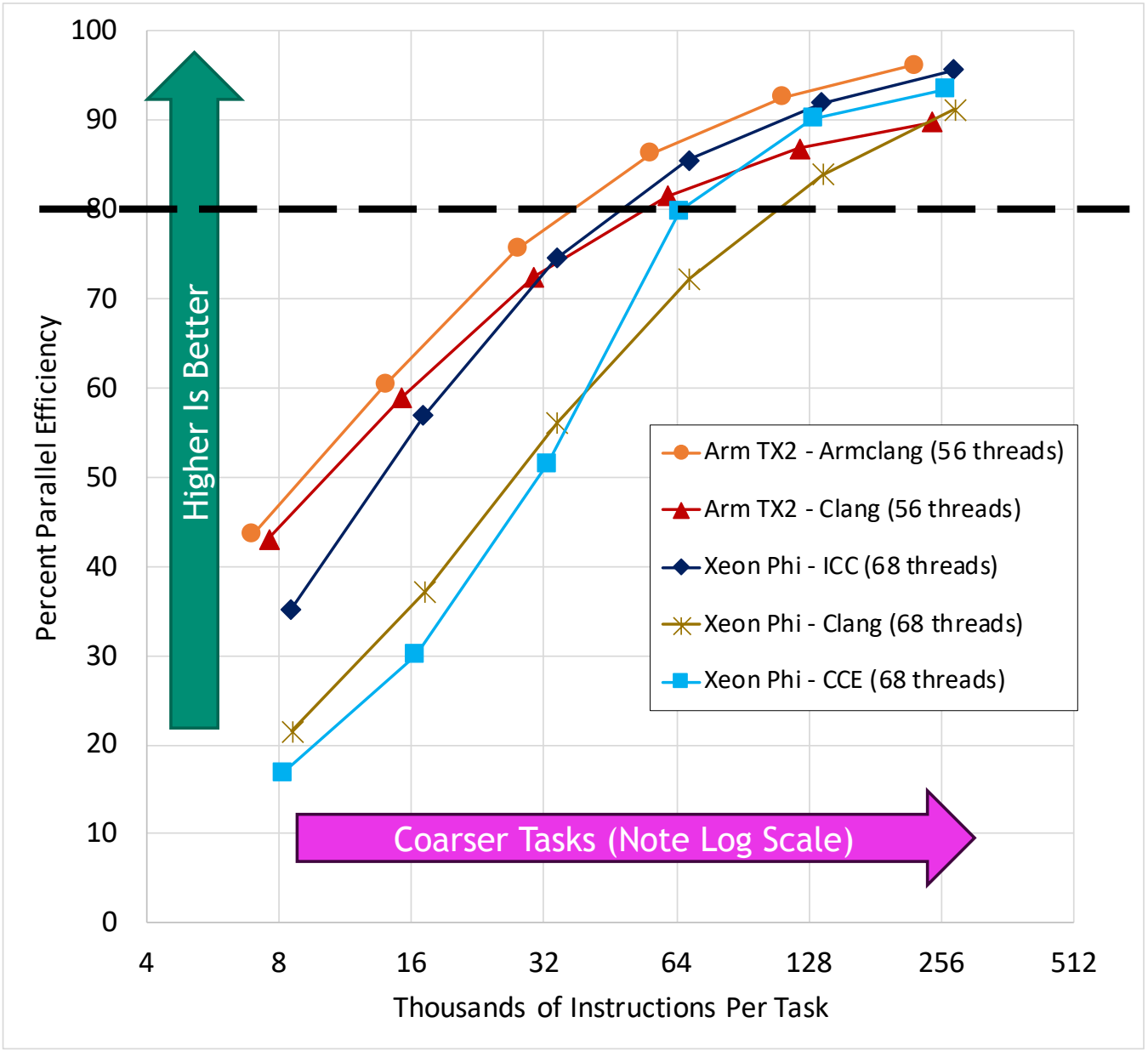
Parallel Efficiency on Intel Xeon & IBM P9 (One Thread/Core)



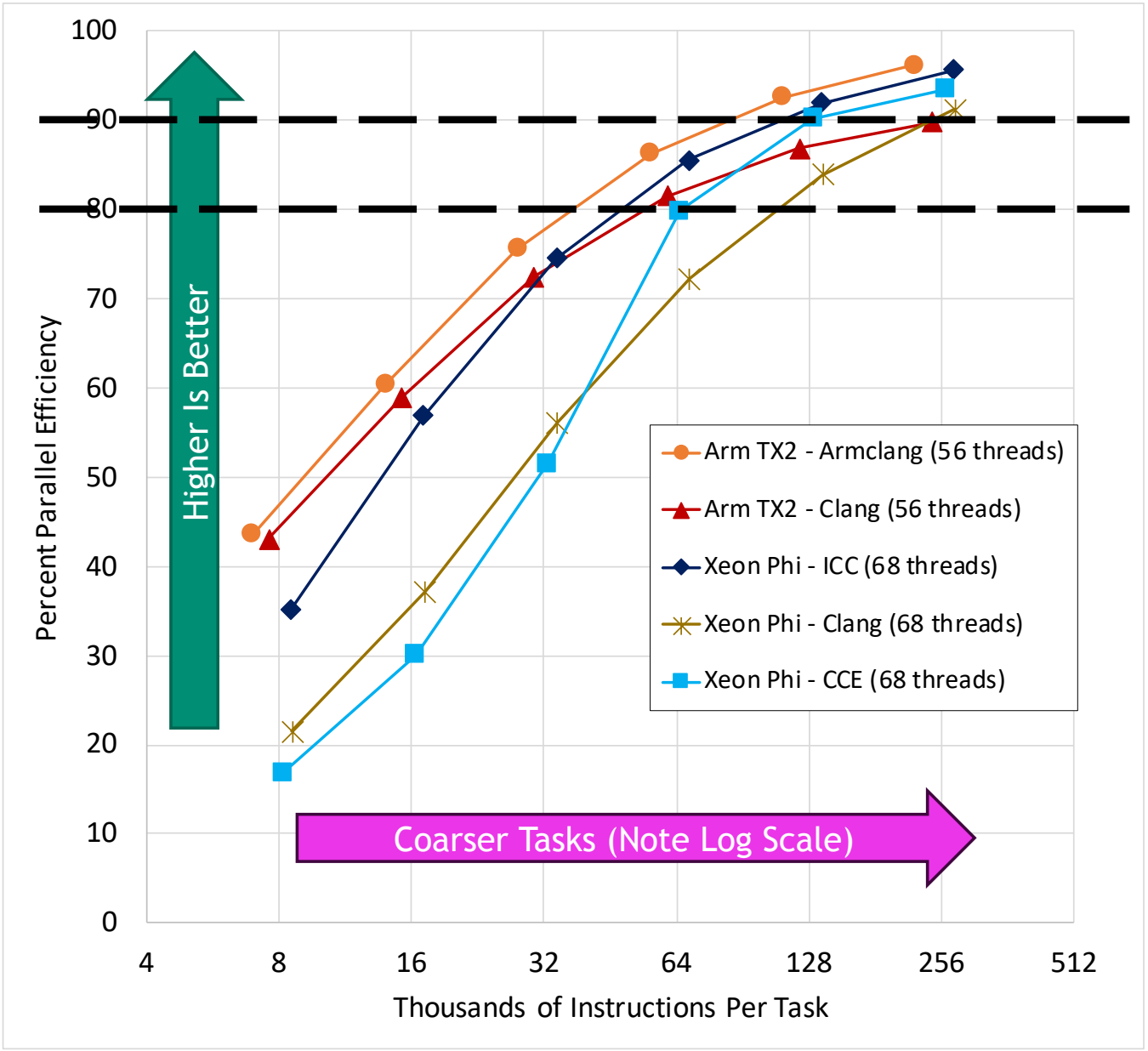
Parallel Efficiency on Arm TX2 & Xeon Phi (One Thread/Core)



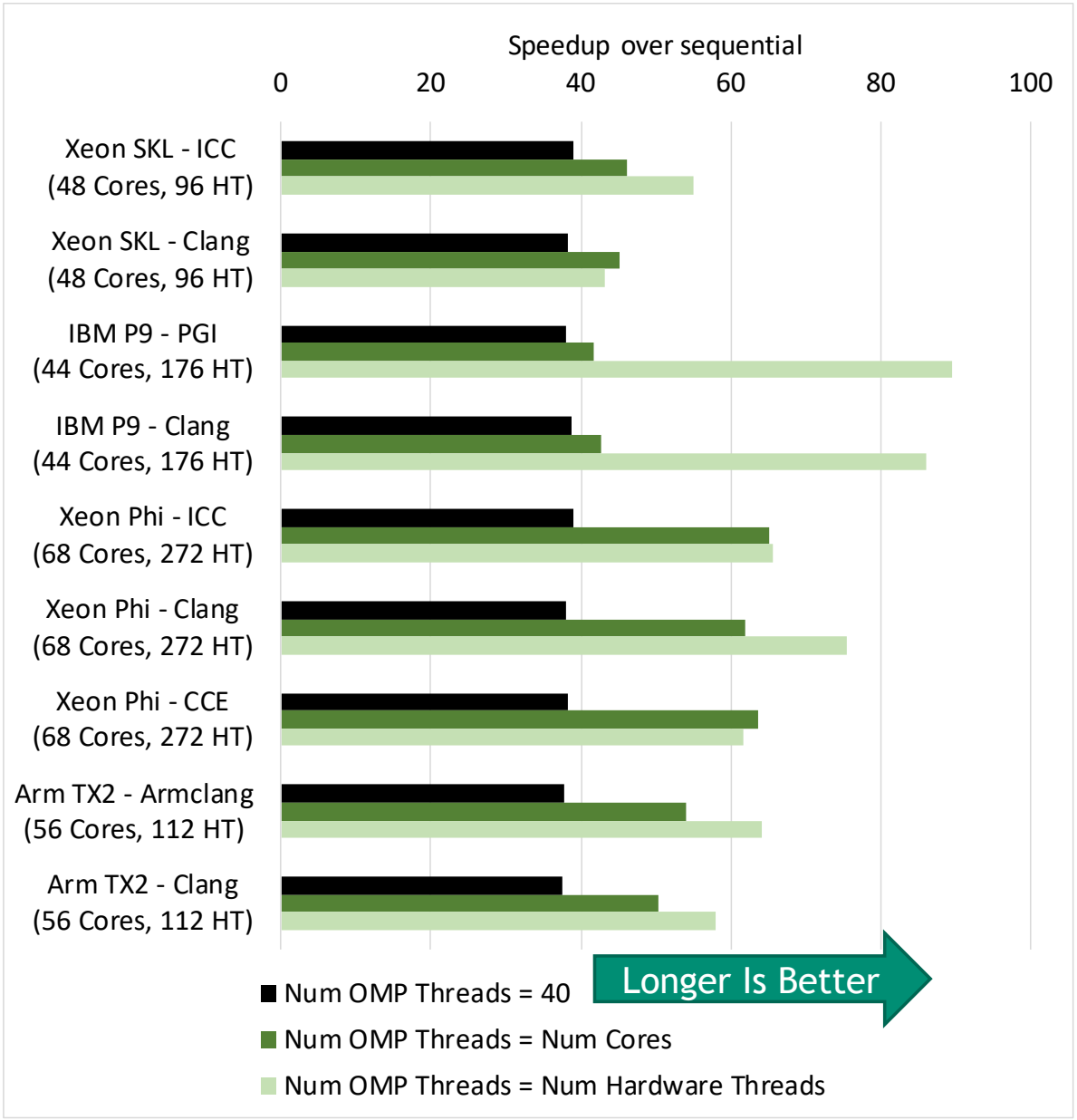
Parallel Efficiency on Arm TX2 & Xeon Phi (One Thread/Core)



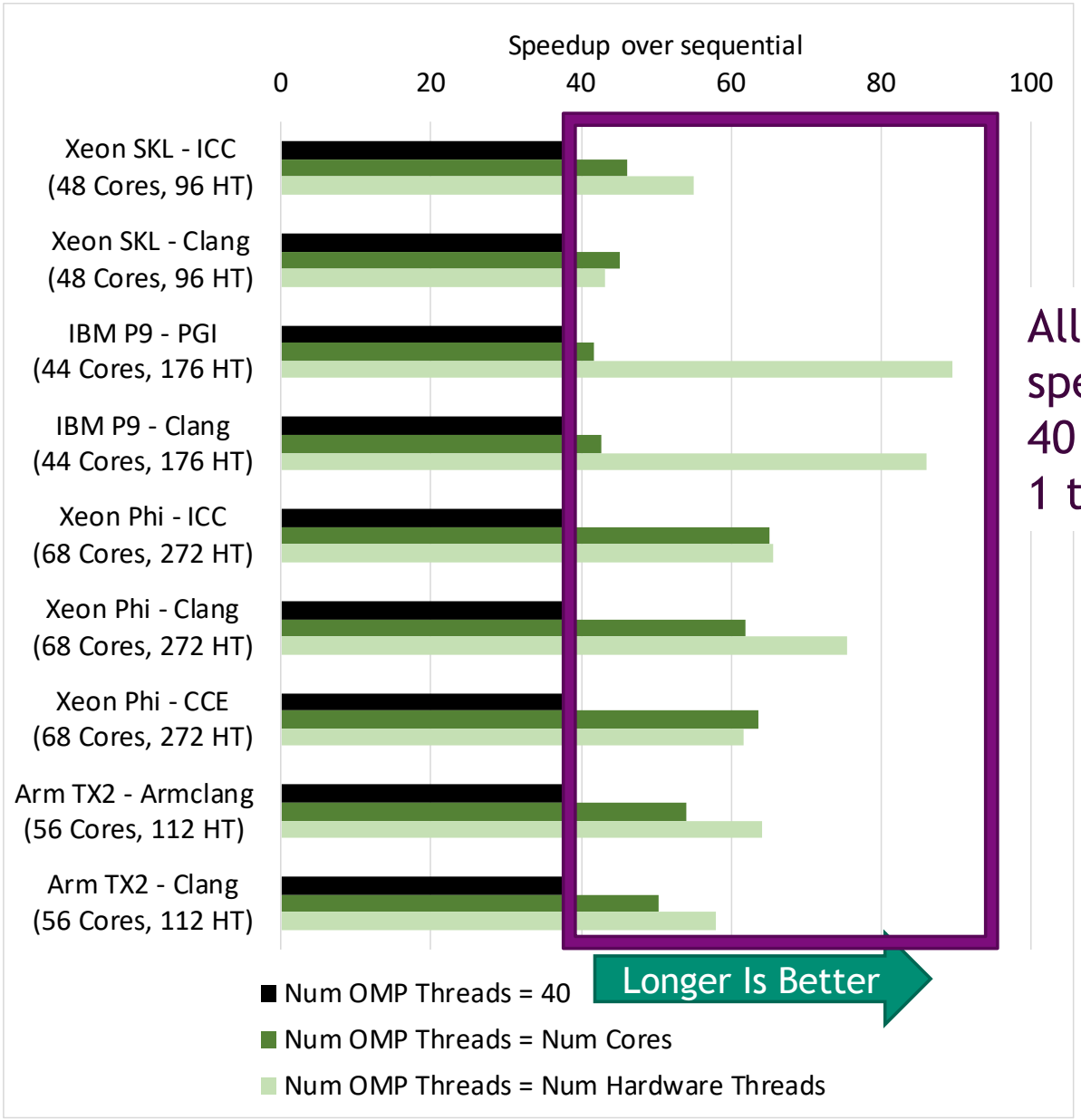
Parallel Efficiency on Arm TX2 & Xeon Phi (One Thread/Core)



Speedup on Coarsest Problem: SMT Usefulness Varies by System

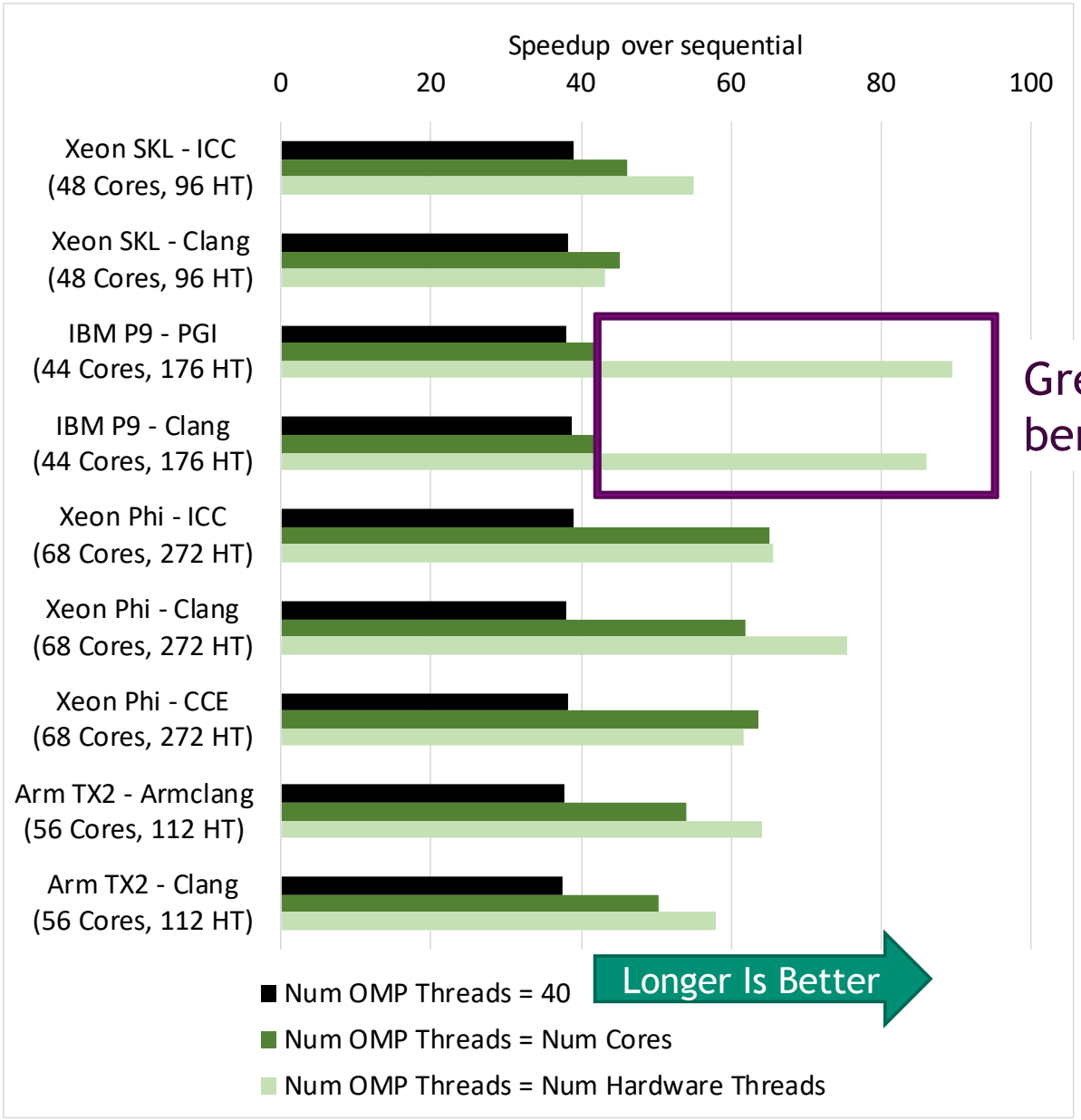


Speedup on Coarsest Problem: SMT Usefulness Varies by System



All improve speedup from 40 threads to 1 thread/core

Speedup on Coarsest Problem: SMT Usefulness Varies by System



Load Balancing Metric: Child Tasks Moved Per Thread Per Second

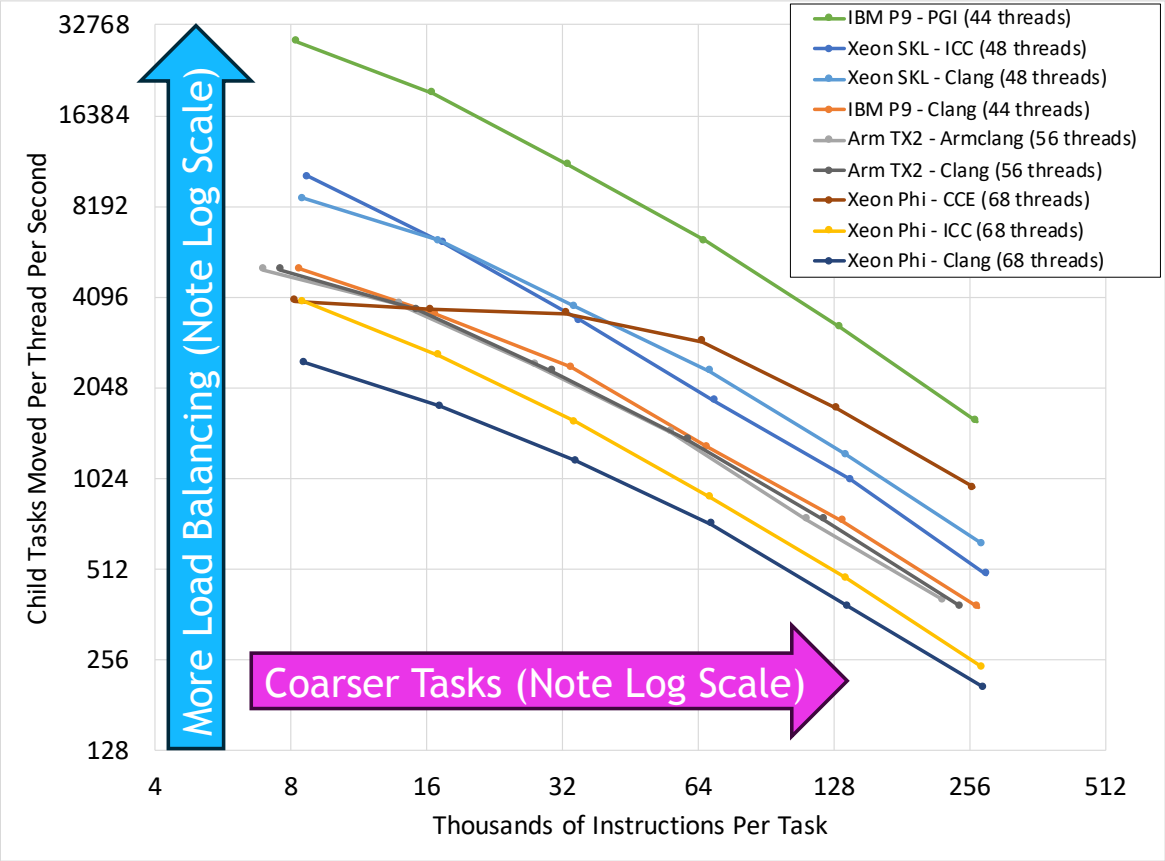


Table 3. Pearson correlation between speedup and number of moved child tasks per second per thread

SHA-1 ops. per task	5	10	20	40	80	160
Pearson correlation	0.69	0.59	0.42	0.42	0.38	0.12

Load Balancing Metric: Child Tasks Moved Per Thread Per Second

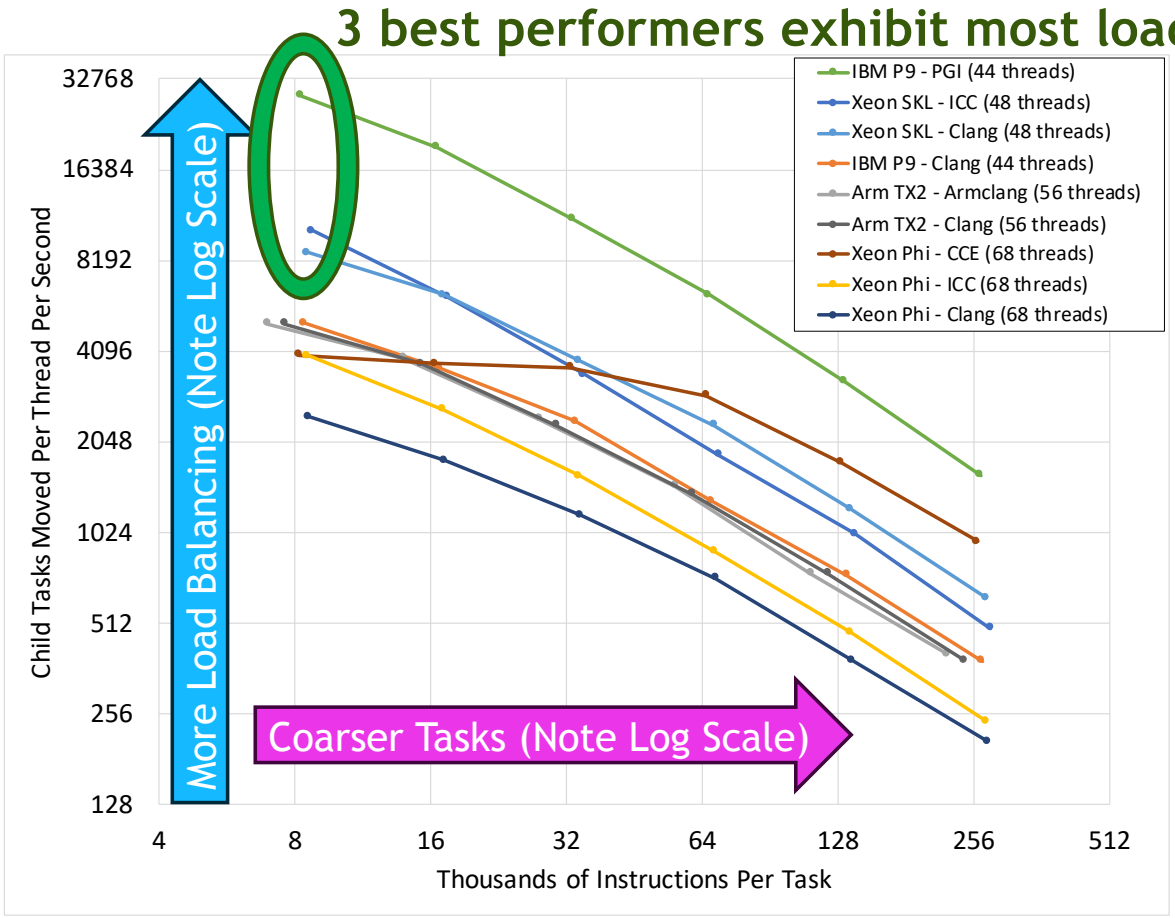


Table 3. Pearson correlation between speedup and number of moved child tasks per second per thread

SHA-1 ops. per task	5	10	20	40	80	160
Pearson correlation	0.69	0.59	0.42	0.42	0.38	0.12

Load Balancing Metric: Child Tasks Moved Per Thread Per Second

3 best performers exhibit most load balancing

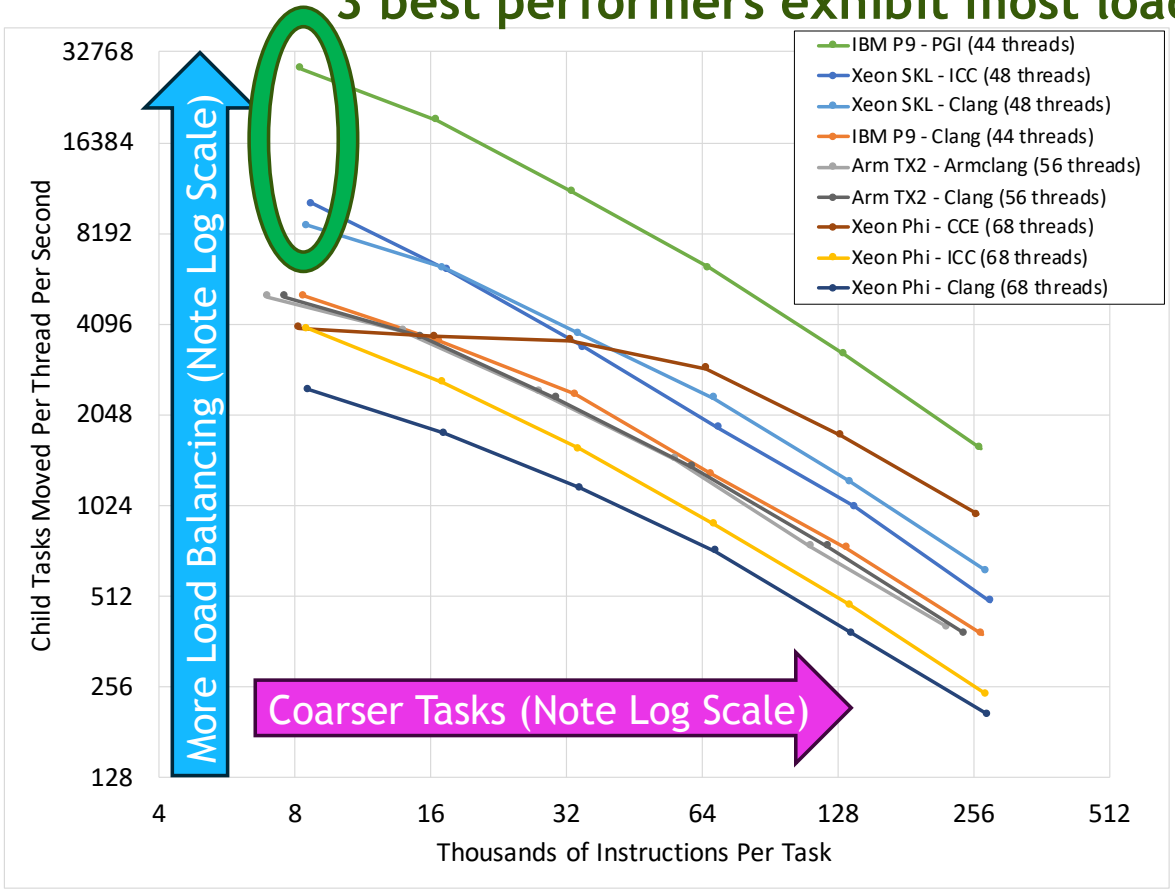


Table 3. Pearson correlation between speedup and number of moved child tasks per second per thread

SHA-1 ops. per task	5	10	20	40	80	160
Pearson correlation	0.69	0.59	0.42	0.42	0.38	0.12

Load balancing important for fine-grained tasks

Conclusion and Updates

Fear not the use of OpenMP tasks if tasks aren't "too small"

- All implementations efficiently handling tasks of *O(100k)* instruction granularity
- Some (vendor) implementations efficiently handling tasks of *O(10k)* instruction granularity
- Clang/LLVM consistently adequate on diverse architectures

New since the paper went to print…

- Clang/LLVM 11 Release Candidate 2 available, with final release imminent
- Support for task reductions on orphaned tasks tested and confirmed
- Will allow future work testing UTS version using task reductions