Analytical Performance Modeling and Validation of Intel's Xeon Phi Architecture

Sudheer Chunduri[†], Prasanna Balaprakash[†], Vitali Morozov[†], Venkatram Vishwanath[†],

Kalyan Kumaran[†]

Argonne National Laboratory[†]
{sudheer, pbalapra, morozov, venkat, kumaran}@anl.gov
Argonne National Laboratory
Lemont, IL 60439

ABSTRACT

Modeling the performance of scientific applications on emerging hardware plays a central role in achieving extreme-scale computing goals. Analytical models that capture the interaction between applications and hardware characteristics are attractive because even a reasonably accurate model can be useful for performance tuning before the hardware is made available. In this paper, we develop a hardware model for Intel's second-generation Xeon Phi architecture code-named Knights Landing (KNL) for the SKOPE framework. We validate the KNL hardware model by projecting the performance of minibenchmarks and application kernels. The results show that our KNL model can project the performance with prediction errors of 10% to 20%. The hardware model also provides informative recommendations for code transformations and tuning.

KEYWORDS

KNL, performance, projection, benchmark, analytical modeling

1 INTRODUCTION

Evaluating the performance of scientific applications on a wide range of hardware plays a central role in many areas of high-performance computing. This is challenging and time-consuming since it requires significant human effort. The difficulty is exacerbated by the constantly growing scale of complexity in the hardware design and adoption of new hardware-specific algorithms for the scientific applications. Software development cycles have become relatively long, hindering scientific productivity. Furthermore, studying the performance of scientific applications on emerging architectures is even challenging when they are not yet available.

Performance modeling is one the most important approaches in performance evaluation. The models are used to capture various facets of an application on a target system. Approaches for modeling the performance range from discrete-event simulations to queuing models. Analytical modeling is encapsulating varying degrees of interactions between applications, system software, and architecture by using closed-form expressions for predicting performance metrics. Performance profiling tools, (e.g., TAU [7], HPCToolkit [5]) focus mostly on the performance characteristics of a given implementation. These tools rely on the actual execution of the workload on the hardware. Architecture simulators are able to reveal performance responses of various hardware configurations, but they treat workloads as black boxes and are time consuming. Application performance models [3, 9] summarize the asymptotic performance characteristics. They can estimate performance bounds at a coarse granularity; however, they express only the characteristics of a given implementation without capturing the internal relationship between the control and the data flow.

Recently, performance projection frameworks have been used to reduce the amount of human effort and to streamline performance projection process [6, 8]. They try to overcome the limitations of general-purpose performance tools and adhoc modeling practices. These frameworks abstract the workload's behavioral properties and the hardware characteristics and combine both to project performance. These properties are data flow, control flow, computation intensity, concurrency, and communication patterns. The hardware characteristics are available execution units, instruction set, order of executions, memory hierarchy, network, and I/O.

In this paper, we focus on the SKOPE [1] performance projection framework. Given a formalized description of the workload's performance behavior, SKOPE automatically analyzes, tunes, and projects the workload's performance for a given parameterized target hardware. The frontend of SKOPE is a *code skeletons* language, a uniform description of the semantic behavior of a workload. According to the semantics and the structures in the code skeleton, the backend explores various transformations, synthesizes performance characteristics of each transformation, and evaluates the transformation with various hardware models.

The SKOPE hardware models perform incremental instruction scheduling. This was sufficient to model in-order architectures such IBM Power A2 or Nvidia GPUs; however, most current-generation processors schedule instructions out of order and execute them using multiple, not necessarily symmetric, instruction pipelines. In this paper, we present the SKOPE extension to model these architectural features. We will present the KNL hardware model as a case study for validation. Our key contributions are:

- We extend the SKOPE language definition and its parser to enable data dependency specification at the control statement level.
- We develop a data dependency analysis and a scheduling algorithm. The analysis produces a dependency graph by identifying

write-after-read, read-after-write, and write-after-write dependecies. The scheduling algorithm adopts a critical path algorithm graph and schedules the instructions on mutiple pipelines.
- We develop a KNL analytical hardware model using the publicly available data on KNL processor.
- We demonstrate the effectiveness of the new models by projecting the performance of minibenchmarks and of microkernels. We achieved prediction accuracy between 80% and 90%.
- We describe the procedure to measure the latency at the granularity of a few cycles, which can be used on other architectures.

The paper is organized as follows. An overview of SKOPE and the extensions to the SKOPE hardware model are described in Section 2. In Section 3, the KNL-specific parameters used for the projection are discussed and the hardware model is validated by comparing the performance on the hardware. In Section 4, we summarize the work and then discuss future work.

2 BENCHMARKS AND SKOPE EXTENSIONS FOR KNL

The HACC framework uses N-body techniques to simulate the formation of structures under the influence of gravity in universe. HACCmk is a key routine that calculates the particle force with an O(N^2) algorithm. The source code and the skeleton of the HACCmk are shown in Listings 1 and 2 repsectively. Nek5000 is a high-order, incompressible Navier-Stokes solver based on the spectral element method. For our study, we use the two kernels *mxf12* and *glsc3i* from Nekbone, a simplified version of Nek5000 These are matrix multiplication with a 12x12 size inner product and vector dot product kernels, respectively.

Listing 1: `HACCmk`**'s source code.**

```
void Step10()
{
 const float ma0, ma1, ma2, ma3, ma4, ma5;
 float dxc, dyc, dzc, m, r2, f;
 float xi, yi, zi;
 int j;
 xi = 0.; yi = 0.; zi = 0.;
 for ( j = 0; j < count1; j++ )
 {
  dxc = xx1[j] - xxi;
  dyc = yy1[j] - yyi;
  dzc = zz1[j] - zzi;
  r2 = dxc * dxc + dyc * dyc + dzc * dzc;

  f = pow(r2 + mp_rsm2, -1.5) \\
      -( ma0 + r2*(ma1 + r2*(ma2 \\
      + r2*(ma4 + r2*(ma4 + r2*ma5)))));

  m = ( r2 < fsrrmax2 ) ? mass1[j] : 0.0f;
  f = ( r2 > 0.0f ) ? m * f : 0.0f;
  xi = xi + f * dxc;
  yi = yi + f * dyc;
  zi = zi + f * dzc;
 }
 *dxi = xi;
 *dyi = yi;
 *dzi = zi;
}
```

Listing 2: `HACCmk`**'s code skeleton.**

```
def main ()
{
  :count = 327
  :nt = 4000
  for i = 0:count
  {
      forall j = 0:nt
      {
        ld xx1
        ld yy1
        ld zz1
        fp dxc: xx1, xxi
        fp dyc: yy1, yyi
        fp dzc: zz1, zzi
        fp r2: dzc
        fma r2: r2, dyc
        fma r2: r2, dxc
        fp f: r2, mp_rsm2
        fp f1: f
        fp f: f1, f
        fp f: f // 1/(sqrt(f*f*f))
        fma r1: r2, ma4, ma5
        fma r1: r1, r2, ma3
        fma r1: r1, r2, ma2
        fma r1: r1, r2, ma1
        fma r1: r1, r2, ma0
        fp f: f, r1
        ld m
        fp f: f, m
        fp m1: r2, fsr
        fp f: z, f, m1
        fp m2: r2, z
        fp f: z, f, m2
        fma xi: xi, f, dxc
        fma yi: yi, f, dyc
        fma zi: zi, f, dzc
      }
  }
}
```

2.1 SKOPE

SKOPE is a performance projection framework. Given a formalized description of the workload's performance behavior, SKOPE automatically analyzes, tunes, and projects the workload's performance for a target hardware. The SKOPE language [1] is the front-end of the framework. Its syntax allows the modeler to specify how input data may affect the control and data flow. The key aspect of the SKOPE language is to express *what* the workload *needs to do* algorithmically, without specifying *how* it is done in the current implementation. The resulting description, referred to as a *code skeleton*. The SKOPE framework has been used to model not only parallel applications and parallel architectures [2] but also distributed workflows [4].

To model KNL architecture that uses out-of-order execution on multiple instruction pipelines, we have extended SKOPE with data dependency analysis and a scheduling algorithm.

The dependency analysis module checks for read-after-write, write-after-read, write-after-write dependencies. In `fp`, `xp`, and `fma` statements, the dependencies are expressed as 3-tuples (dependent variable, **":"**, a list of independent variables). For each BST, based on the dependencies, the module creates a directed acyclic graph G, where the nodes and edges represent statements and their dependencies. The module starts with a graph G, where each node n_i represents a statement s_i, and processes the statements in the BST in order. For each variable v_j in s_j, the immediate previous statement s_i is found, where v_j is used as a dependent variable. A directed edge e_{ij} from s_i to s_j is created. Figure 1 illustrates the dependency graph for the HACCmk.

For out-of-order scheduling with multiple pipelines, we use the dependency analysis graph G, the latency of each node n_i in G, and

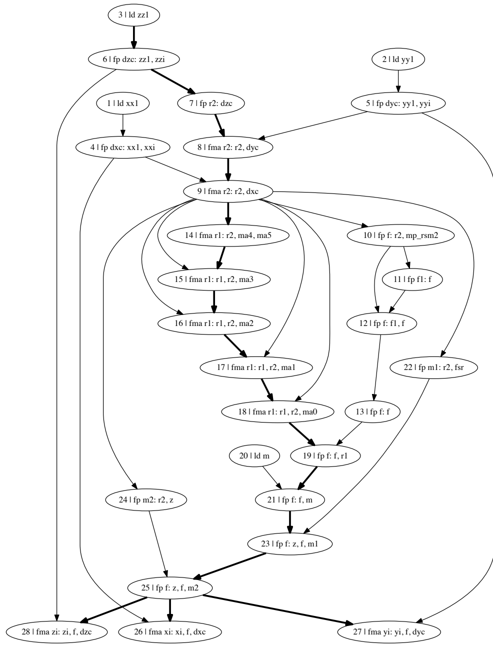

Figure 1: Date dependency graph for HACCmk.

the number of execution units on the hardware. First, unweighted G is converted to weighted G by assigning a weight w_{ij} to edge e_{ij} equal to the latency of execution the node n_i. The source n_i and the destination n_j of an edge e_{ij} are called the parent node and the child node, respectively; a node without a parent is called an entry node; a node without a child is called an exit node. A node cannot start execution before all parent nodes have finished execution. The objective is to assign the nodes of G to the execution units such that the total schedule time is minimized without violating the dependencies. A schedule is efficient if the execution units are used without idle cycles. We adopt critical path and list scheduling heuristics on G for out-of-order scheduling on multiple pipelines.

3 VALIDATION ON THE KNL HARDWARE

We use the KNL hardware model in SKOPE to project the performance of compute kernels. The first challenge involves validating the results when KNL is not yet available. Since the architecture simulator is not public, we validated the model on the KNL hardware once it became available. The second challenge is related to the accurate benchmarking. Since a singe call of a compute kernel consumes a few thousand cycles, accurately measuring the time is important. We developed a low-overhead microbenchmarking methodology to accurately measure the cycle and instruction counts. We validated the hardware model using microbenchmarks first and then using the HACCmk and Nekbone compute kernels.

3.1 HACCmk and Nekbone Kernel Validation

We use the code skeletons of the HACCmk and Nekbone kernels as input to the KNL hardware model to derive performance projections. We validate the projected performance by running these benchmarks on the KNL hardware.

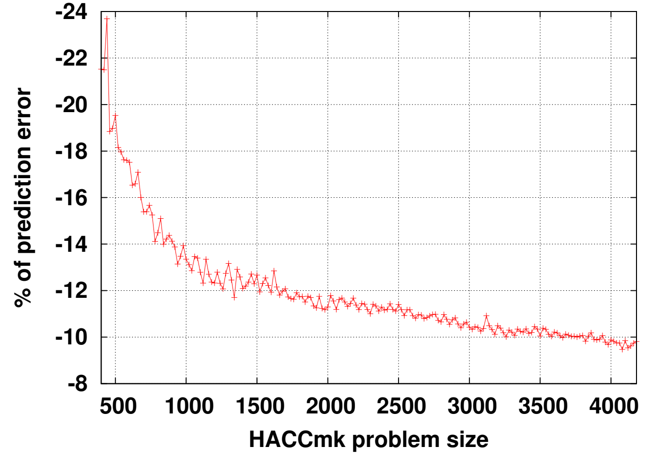

Figure 2: Percentage of prediction error for the HACCmk with different problem sizes

For a given code skeleton, the abstract execution model in SKOPE experiments with different code transformations and picks the best code transformation heuristically. One such code transformation is loop unrolling. Table 1 shows the projected cycles for the HACCmk microkernel with three input sizes using different unroll factors from 1 to 16. The hardware model picks four as the best unroll factor based on the analysis of number of stalled cycles in the execution and the number of data elements (representing the registers on the hardware) saved. Next, we compare the cycles projected by the hardware model using the unroll factor of four with the measurement on the hardware.

Table 1: HACCmk - projection of execuyion cycles

Code transform.	N=2000	N=3000	N=4180
Unroll:1	3678750	5518125	7688587
Unroll:2	1859812	2789718	3887008
Unroll:4	**960562**	**1440843**	**2007575**
Unroll:8	608015	912023	1270752
Unroll:16	585023	877535	1222698

Table 2: HACCmk - Projected vs. Measured cycles

	N=2000	N=3000	N=4180
Unroll:4	960562	1440843	2007575
Cycles on hardware	1149040	1727047	2407385
% of error	-16.40	-10.42	-9.81

Table 2 shows the cycles measured on the hardware and the corresponding prediction error. In Figure 2, we show the percentage of prediction error as a function of the problem size. We observe that the error starts 21.5% for a smaller problem size to 9.8% for a larger problem size. These reasonably high prediction accuracies validate the KNL hardware model, where the unroll factor predicted by the hardware model concurs with the loop unroll factor selected by the compiler/hardware.

The model extensions discussed in Section 2 focus primarily on the instruction scheduling and out-of-order execuction. Intuitively, this information should be sufficient to project the performance with reasonable accuracy for compute-intensive kernels such as HACCmk. The primitive memory model that is part of the hardware model would be crucial to the memory-intensive kernels such as the Nekbone kernels. The memory model accounts for the latency of a memory operation for the first touch, and it accounts for the locality in the stride-1 memory access streams.

We now use the hardware model to project the performance of the Nekbone microkernels *mxf12* and *glsc3i*. Tables 3 and 4 respectively show the prediction errors for the projection of kernels *mxf12* and *glsc3i* as -10.33% and -8.30%. The hardware model was able to project the performance well even for these kernels. These reasonably good prediction accuracies validate the memory modeling aspects of the hardware model.

Table 3 shows the validation for the *mxf12* kernel with three matrix sizes. While the prediction error for the smallest matrix size is relatively high compared with the bigger matrix sizes, the overall prediction accuracy is reasonably good. Also, the instructions projected by the hardware model match closely with the instructions retired as measured on the hardware. As shown in Table 4, the hardware model selects 4 as the best unroll factor for the *glsc3i* kernel as well.

Table 3: *mxf12* - Projection of number of cycles

mxf12 Size	Cycles Measured	Cycles Projected	Instructions Measured	Instructions Projected
(144, 12, 12)	23582	21456	42642	44352
(12, 12, 12)	2223	1788	3567	3696
(12, 144, 12)	22425	20004	40001	41712
Total	**48230**	**43248**	**86210**	**89760**
%of error:	-	-10.33	-	4.12

Table 4: *glsc3i* - Projection of number of cycles

Code Transform	Cycles Projected	Cycles Measured	Reference Cycles	Instructions Measured
unroll:1	18579456	**5309343**	**4930055**	**5308435**
unroll:2	9289728			
unroll:4	**4866048**			
unroll:8	2875392			

4 CONCLUSION AND FUTURE WORK

We have developed a hardware model for the second-generation Intel Xeon Phi architecture code named Knights Landing, by extending the SKOPE execution model to support out-of-order instruction scheduling and pipelining. The model was used to project the performance of HACCmk and Nekbone kernels that are derived from critical regions of two exascale applications, HACC and Nek5000, respectively and was validated by using the runs on the hardware. The model can be used to project application performance and suggest effective code transformations on the target hardware even before the production runs. The model can help performance engineers and hardware designers set performance-tuning goals, select code optimizations, and, above all, identify which hardware features are more suitable for their applications. This information can help them advocate for certain proposed hardware features by the vendors in future microarchitectures such as Knights Hill.

Future work includes extending the SKOPE framework to support a wider selection of hardware models so that it can be used to study various computer architectures. We also plan to extend the skeleton language to model applications with a specific focus on data-flow-based optimizations, asymmetric pipelines, and more code transformations such as loop splitting and loop fusion.

Acknowledgments We thank the ALCF application and operations support staff for their help. This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357 and the RAMSES project under the Next Generation Networking for Science Program.

REFERENCES

[1] J. Meng, X. Wu, V. Morozov, V. Vishwanath, K. Kumaran, and Valerie Taylor. 2014. SKOPE: A framework for modeling and exploring workload behavior. In *11th ACM Conference on Computing Frontiers*.

[2] Jichi, Guo and Jiayuan, Meng and Qing, Yi and Vitali, Morozov and Kalyan, Kumaran. 2014. Analytically modeling application execution for software-hardware co-design. In *IEEE 28th International Parallel & Distributed Processing Symposium*.

[3] Darren J. Kerbyson, Henry J. Alme, Adolfy Hoisie, Fabrizio Petrini, Harvey J. Wasserman, and M. Gittings. 2001. Predictive performance and scalability modeling of a large-scale application. In *SC*.

[4] Maheshwari Ketan, Jung Eun-Sung, Meng Jiayuan, Vishwanath Venkatram, and Kettimuthu Rajkumar. 2016. Improving multisite workflow performance using model-based scheduling. In *Future Generation Computer Systems*.

[5] L. Adhianto and S. Banerjee and M. Fagan and M. Krentel and G. Marin and J. Mellor-Crummey. 2010. HPCToolkit: Tools for performance analysis of optimized parallel programs.. In *Concurr. Comput.*

[6] Meng, J. and Morozov, V. A. and Kumaran, K. and Vishwanath, V. and Uram, T. D. 2011. GROPHECY: GPU performance projection from CPU code skeletons. In *SC*.

[7] Sameer S., Shende and Allen D., Malony. 2006. The TAU parallel performance system.. In *International Journal of High Performance Computing Applications*. 20, no.2,287–311.

[8] Sameh, Sharkawi and Don, DeSota and Raj, Panda and Stephen, Stevens and Valerie, Taylor and Xingfu, Wu. 2012. SWAPP: A framework for performance projections of HPC applications. In *IEEE IPDPS2012 Workshop on Large-Scale Parallel Processing*.

[9] K. L. Spafford and J. S. Vetter. 2012. Aspen - A domain specific language for performance modeling. In *SC*.