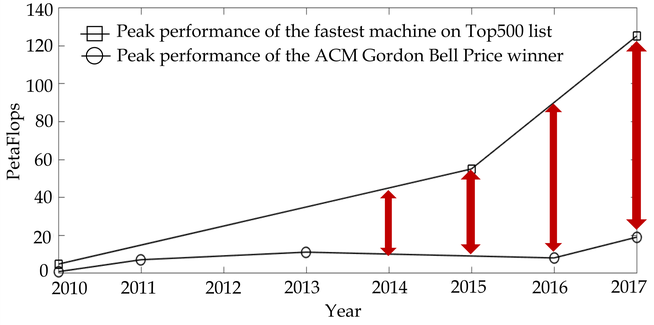
Understanding potential performance issues using resource-based alongside time models

Nan Ding[1], Victor W Lee[2], Wei Xue[3], Weimin Zheng[3]

1. Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA 2. Intel Corporation, Santa Clara, CA 95054, USA 3. Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

Motivation

The gap between actual and the expected performance is increasingly widened due to the growing complexities of both machines and scientific applications. To bridge the gap, performance analysis has been considered as a necessary step, and performance analysis tools are becoming one of the most critical component in today's HPC systems. Performance modeling, the core technology to identify key performance characteristics and predict potential performance bottlenecks, is becoming an indispensable tool to understand the performance behaviors and guide performance optimizations of HPC applications.



Contributions

- Hardware counter-assisted profiling to identify the key kernels and non-scalable kernels in the application
- Resource-based Alongside Time (RAT) model
  - understanding the potential performance issues
  - predicting performance in the regimes of interest to developers and performance analysts
- Easy-to-use performance modeling tools for scientists and performance analytics

Kernel Identification

Take the following functions as kernels:

a) time proportion is larger than the user-defined threshold

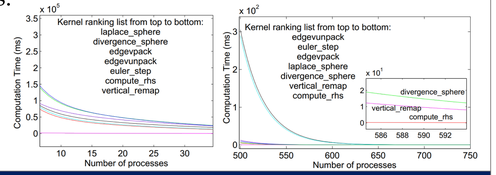
b) consumed time do not decrease in the profiling runs (non-scalable)

c) merge the test function as one kernel

Profiling processes: P=[p_0, p_1, p_2, …, p_n]

Study case: HOMME:

Users focus on optimizing the performance of kernel *compute_rhs* and *euler_step*. However, kernel *edgevunpack* should be taken into account when conducting large scale runs.



Resource-based Alongside Time Model Construction

Total runtime (T_{app}) equals to the accumulation of computation time (T_{comp}), non-overlapped communication time (BF_{comm}*T_{comm}) and the initialization/finalization time (T_{others}).

$$T_{app} = T_{comp} + BF_{comm} * T_{comm} + T_{others}$$

$$T_{comp} = \sum_{i=1}^{K} \left(\frac{\#inst * CPI_{core_i}}{F * P} + BF_{mem_i} * T_{mem_i} \right)$$

waiting time for memory

execution time for instructions

$$BF_{mem_i} = \frac{\#stall_{load_i} + \#stall_{store_i}}{\sum_{m=L1}^{Mem} \#uops_{i_m} * L_m / (F * P)}$$

$$CPI = CPI_{core_i} + \sum_{m=L1}^{Mem} \#miss\ uops_{i_m} * L_m$$

$$T_{comm} = T_{p2p} + T_{coll}$$

$$T_{p2p} = a * S^b + c$$

$$T_{coll} = a * \log(P) + b * S + c$$

How is the model item derived?

From hardware counters

$\#inst$ $\#uops$ $\#stall_{load}$ $\#stall_{store}$ CPI_{core_i}

From benchmarking/Guide

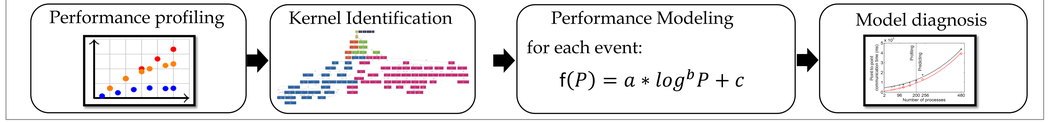
L_m F (CPU frequency)

From PMPI interface

S (communication volume)

User-defined

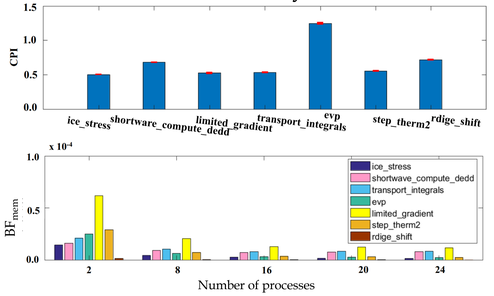
P (number of processes)



Model Diagnosis

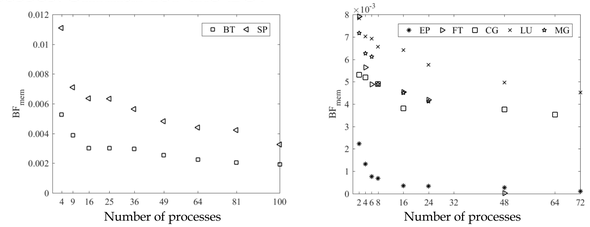
CICE:

Limited_gradient (L) and transport_integrals (T) have similar CPIs. However, L has a higher BF_{mem} than T which indicts that L suffers from a lower memory traffic.



NPB:

SP has a relatively bad memory behavior than BT. By looking into the SP code, it has some non-continuous memory accesses. Similar for the LU.



Comparison: