Say "Yes" to Dumb Operating Systems and Smart Applications

Alexandra Fedorova, *Simon Fraser University, Vancouver Canada*

1 Introduction

The advent of chip multiprocessors (CMP) has spurred research on new OS scheduling algorithms that enable applications to make the most out of CMP systems. While many of these algorithms are very effective, their complexity casts doubts as to their practicality and scalability. I propose a wild and crazy solution to this problem. It involves taking the smarts out of the operating system and putting them into applications, since who knows better about application's architectural properties than applications themselves? In the rest of the proposal I elaborate on this idea.

2 OS Complexity

One class of new scheduling algorithms uses sophisticated analytical modeling to determine how threads should be scheduled or co-scheduled on a multicore processor. Running model code (e.g., a linear regression) in the kernel is ineffective (because floating point operations are usually not allowed) and impractical, because of the code's complexity. Complex code is bug prone and difficult to maintain. Therefore, I doubt that it will ever make it into production OS.

Another class of scheduling algorithms relies on continuous monitoring of application performance by the OS via hardware counters. Analysis of performance trends is used to determine the optimal way of assigning threads to cores. Examples include sampling pairs of co-scheduled applications to determine optimal co-schedules, sampling performance of threads on different kinds of heterogeneous cores to determine the "right" core for the thread. Apart from the fact that continuous monitoring of hardware counters by the OS prevents using the counters by applications, this approach has scalability limitations. As the number of cores increases, the amount of monitoring (and the length of the monitoring period) also increases, usually exponentially with the number of cores. Monitoring of the application must be done at each new program phase. If the monitoring period becomes very long, which is bound to happen on many-core systems, the monitoring will fail to complete within a single program phase, and the resultant data will be insufficient for scheduling decisions.

3 Wild and Crazy Solution

The OS complexity in the new CMP algorithms results from the need to learn how applications use architectural resources and how they respond to variations in the availability of those resources. In the existing solutions, the OS attempts to determine this by itself, via external monitoring. At the same time, who knows more about applications' architectural characteristics than the applications themselves? I propose that applications should inform the OS about their architectural characteristics, and the OS should use that information for scheduling. As a result, there will be no need for online monitoring; analytical modeling, if any, will become less complex; and thousand-core scalability will be trivial to achieve. The key ideas in the proposed solution are (a) how to learn and represent applications' architectural characteristics and (b) how to make good scheduling decisions given those characteristics.

Learning the architectural characteristics will be done as part of the application development process, similarly to static feedback-directed optimizations. Applications will be profiled, relevant profiling data will be collected and summarized in an *architectural signature.* This signature will be embedded in the application binary. The signature will contain at least the following information: (1) cache reuse or stack-distance profile, (2) instruction type frequency, and (3) branch prediction and branch frequency statistics. The first item will be used to determine good co-schedules on CMPs with shared caches. On heterogeneous CMPs it will be used to determine the "right" core for the application. For example, if the application has poor cache reuse it is probably memory bound and thus should be run on a low-frequency single-issue core as opposed to a high-frequency super-scalar core. Further, cache reuse profile can help determine the sufficient cache size for the application, so it can be scheduled on the core with enough cache. The second item can be used to determine if the application is floating-point intensive: in the heterogeneous system, the OS will know to run it on a core with sufficient FPU resources. The third item can help predict the degree of ILP achievable in the application. Poor branch predictability and high branch frequency would suggest low ILP. A low-ILP application would be scheduled on single-issue core, while a high-ILP application on a super-scalar OOO core.

To use architectural signatures for optimal scheduling on heterogeneous CMPs, the OS will classify the machine's cores according to issue width, cache size, clock frequency, and presence of FPUs. Then it will match applications architectural signatures with core classification to find a good assignment. To use architectural signatures for finding optimal co-schedules (with respect to cache contention), the OS will analyze cache reuse profiles to determine how groups of applications interact in the cache [1].

The wildness of this idea is due to two factors: First is the suggestion that this simple scheme, involving only small architectural signatures and only coarse-grained classification of cores, will actually improve performance similarly to complex solutions. The second is the expectation that this solution can be made microarchitecture independent, i.e., architectural signatures and classification of cores will be represented with no microarchitectural details. (Microarchitecture independence is desired for OS scalability.)

The future will tell whether this wild and crazy idea will ever become practical and reasonable.

[1] D. Chandra et al. Predicting Inter-Thread Cache Contention on a Multi-Processor Architecture. *HPCA,* 2005