Multi Address Space Parallel Architecture

Sergey N. Zheltov
Software Solutions Group, Intel Corp.
Sergey.Zheltov@intel.com

Stanislav V. Bratanov
Software Solutions Group, Intel Corp.
Stanislav.Bratanov@intel.com

1. PROBLEM DESCRIPTION

Performance is the crucial characteristic of a computational system. Most of current computer architectures suffer from inherent flaws that may impede the overall system performance increase even though their particular components become faster.

For example, higher processor clock frequencies require higher memory response rates to establish the system performance growth. This is currently solved by multiple levels of cache hierarchy with different access latencies and associativity types; at the same time, it is quite difficult to provide a universal cache usage strategy, suitable for any computational task, especially when a cache system is supposed to function transparently to software. That is why most modern architectures provide additional software control features to optimize the cache utilization, namely: hints to load/store instructions on which cache level should be affected by the load/store operation; cache line locks, and hardware data pre-fetch operations [1].

Another problem that affects the performance of modern computer systems is the underutilization of execution units. Most modern general-purpose processors are designed for parallel (explicit or super-scalar) execution by dedicating specialized execution units to executing a set of specific instructions. An ordinary program code generated by a high level language compiler usually employs a limited subset of execution units (normally an integer arithmetic unit – ALU) and occasionally loads other units, such as floating point or a unit for vector operations. This results in constant operation of some units while the others are periodically stalled.

The underutilization issue is to some extent solved by Symmetric Multi-Threading systems that execute more than one program thread on a physical processor, thus allowing a thread to use free execution resources [2]. But still, if the program threads are not properly designed, execution stalls may occur when all threads demand the same execution resource [3].

All the above testify the need of more developed software control over the program execution and execution resource utilization.

2. CONCEPT OVERVIEW

The proposed system is supposed to solve many issues that affect performance and other characteristics of current architectures.

The general concept of the newly suggested architecture is to be fully controlled by software.

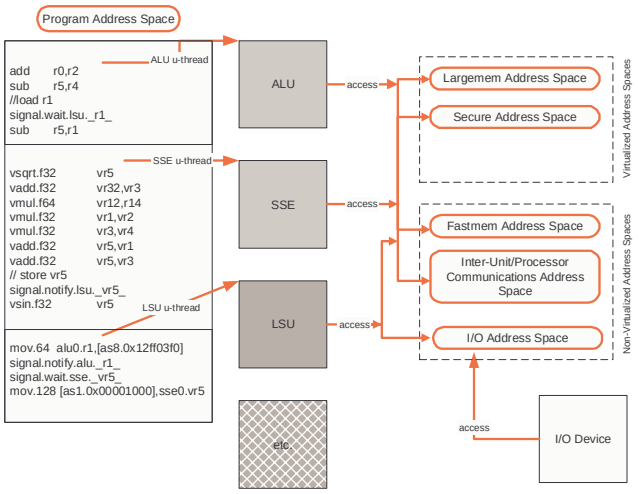
According to the architecture's design, all system resources are represented by a set of execution units and address spaces, the latter, in tum, have a set of properties assigned to each. The properties include: addressing size, virtualization, protection, performance, serialization, and other attributes.

A program may control all parameters of the system in terms and by means of resource allocation and address space accesses. System buses and peripheral devices are addressed via a special address space.

A special space is dedicated to inter execution unit communications.

The operating system owns a system control address space. This address space contains special structures to maintain efficient execution, thread/process separation, debugging capability, resource sharing, virtualization and protection.

A new micro-threaded programming model is introduced to facilitate parallel utilization of execution units.



3. ADVANTAGES

Based on the above concept it becomes possible to avoid cache memory pollution providing fast memory addressed and managed by software.

The utilization of execution units may be also increased as their work is planned for each particular task on a micro-thread basis.

Careful planning of the type of memory (or input/output) operations and the amount of memory to be consumed reduces the bandwidth each memory/storage device is required to support.

Besides, introducing separate address spaces and inter-space data move operations helps extend the basic architecture and adapt it to a particular task and/or environment.

Moreover, as execution units are not naturally homogeneous, that is, require different data types, memory storage characteristics and different execution times, it may appear to be of avail to group the execution units closer to the address spaces that contain resources most probable to be accessed by such units. This may help solving problem of power concentration and distribution over a die.

At last, having system management, I/O, and inter-processor synchronization performed by separate units within separate address spaces will efficiently hide the latencies (compared with thousands of cycles for I/O operations and inter-processor synchronization in, for instance, NUMA systems).

4. REFERENCES

[1] IA-32 Intel® Architecture Software Developer's Manual. Volumes 1-3. Intel Corp., 2004

[2] D.Marr et al. "Hyper-Threading Technology Microarchitecture and Performance". *Intel Technology Journal, Q1 2002*.

[3] Y.-K. Chen et al. "Media Applications on Hyper-Threading Technology". *Intel Technology Journal, Q1 2002*.