A Hardware Independent OS Interface for Parallel Software

by Sean Halle, UC Santa Cruz seanhalle@yahoo.com

Designing an OS that exposes no hardware details is a special challenge because an OS is intimately tied to the hardware. But, for parallel software to enjoy the commercial advantages that serial software has benefitted from, it needs to be equally hardware agnostic.

Such an OS must provide all the services required by programs and users. These include the ability to run a program, to install programs and data, to access data from within a program, to communicate from a program to another program, to another machine, or to hardware, and having a security mechanism.

An OS presents, to an application, the interface to the "universe". Everything outside of an application is seen through the OS's abstractions. For the OS to present a HW-implication-free interface to an application, its abstractions must be free of implications about the hardware under the interface and expose no OS-implementation related information.

Such an OS interface is difficult to define because HW implications arise in many places:

1. Implied by the format of primitive data types in data
2. Implied by location-information attached to names
3. Implied by the semantics of the computation model that code is in terms of
4. Embedded within code via OS commands, HW commands, and location-containing names

Isolating an application from these sources of HW implications requires:

1. A data-format boundary around all the applications. Data is only allowed to cross the boundary by passing through a translator. Data is translated to the standard, then to an internal format.
2. Separating pure names from both location information and meta-information. Location is only needed inside a specific OS implementation – it only has meaning within its context. Meanwhile, meta-information is only needed by applications and people, to search for the pure name of the desired data. The OS implementation then translates pure name to location to retrieve the data.
3. An intermediate format for programs. The format must be both HW independent and enable high performance translations to specific hardware.
4. OS commands that are HW implication free.

The elements of the proposed interface that enable its benefits:

1) separating the OS's interface from its implementation(s) – the interface remains the same across hardware, while implementations vary widely.
2) using HW-free abstractions to model the universe for the application – everything is a processor, including files, running programs, and the OS itself.
3) separating pure name from attached information – a name discovery service is used by applications to search for the pure name of the processor holding the desired data. A pure name is a black-box.
4) data format adaptors – an adaptor parses data coming in according to a grammer. Primitive data, such as integers, floats, and chars, is identified in the grammar and translated to the internal primitive format. To cross the boundary, a data-stream must be registered and an adaptor provided. This isolates the hardware-detail-aware portions of applications to just the adaptors.
5) translating code at install – a split compiler is used. First code is compiled to an intermediate format and bundled with application data (and adaptors for that data). Then, during install, the intermediate format is translated to specific machine code. This allows complex re-structuring of code to fit the particular hardware (run-time methods, like JITs, haven't enough time for major re-structuring).
6) a spawned-processor intermediate format – a running program consists of "spawners" that have guards. The guards select sets of input data that pass a boolean test, then the spawner creates a new processor for each data-set. The spawned processor only lives long enough to process the one set of data. It is undefined how many such spawned processors exist at once, and undefined the order of input sets being processed (but fields can be added to data structures and to the boolean to control the order when needed). This reduces the number of constraints on scheduling, thus increasing the freedom of the hardware-specific compiler to make the parallelism-granularity fit the hardware.

[see codetime.sourceforge.net for further discussion and supporting (unpublished) papers]