ΠΑΡΟΥΣΙΑΣΗ
ΔΙΔΑΚΤΟΡΙΚΗΣ ΔΙΑΤΡΙΒΗΣ

ΗΜΕΡΟΜΗΝΙΑ: Πέμπτη, 16 Οκτωβρίου 2014
ΩΡΑ: 12.00
ΑΙΘΟΥΣΑ: Αίθουσα Σεμιναρίων (ισόγειο Ι1-Ι2)
Κτήριο Τμήματος Μηχανικών Η/Υ & Πληροφορικής
ΟΜΙΛΗΤΗΣ: Δημήτριος Σαούγκος

Θέμα

«Mapping Loop-Based Programs onto a Multithreaded Processor»

Επταμελής Εξεταστική Επιτροπή:

1. Γιώργος Μανής, Επίκουρος Καθηγητής του Τμήματος Μηχανικών Η/Υ & Πληροφορικής, Πανεπιστημίου Ιωαννίνων (επιβλέπων)
2. Chris Jesshope, Καθηγητής Πανεπιστημίου του Amsterdam
3. Νικόλαος Παπασπύρου, Αναπληρωτής Καθηγητής του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών Ε.Μ.Π.
4. Βασίλειος Δημακόπουλος, Αναπληρωτής Καθηγητής του Τμήματος Μηχανικών Η/Υ & Πληροφορικής Πανεπιστημίου Ιωαννίνων
5. Αριστείδης Ευθυμίου, Επίκουρος Καθηγητής του Τμήματος Μηχανικών Η/Υ & Πληροφορικής Πανεπιστημίου Ιωαννίνων
6. Απόστολος Ζάρρας, Επίκουρος Καθηγητής του Τμήματος Μηχανικών Η/Υ & Πληροφορικής Πανεπιστημίου Ιωαννίνων
7. Κωνσταντίνος Μαγγούτης, Επίκουρος Καθηγητής του Τμήματος Μηχανικών Η/Υ & Πληροφορικής Πανεπιστημίου Ιωαννίνων
Περίληψη

This work offers some insight into the automatic parallelization of loops by introducing and describing a source-to-source parallelizing compiler developed from scratch called C2μTC/SL. Once basic notions and ideas on the field of automatic parallelization have been introduced, the SVP system is described in great detail. It is a novel proposal on multi-core architectures and is what C2μTC/SL targets as output. The SVP is a novel design for a multi-threaded processor that can be bundled together with an OS-on-chip as part of the chip’s ISA (Instruction Set Architecture). The C2μTC/SL source-to-source compiler is described afterwards: its purpose is to take as input any legacy C code and transform it into a parallel SL program. The compiler’s main target constructs are the loops since a loop is where most of the execution time of an application takes place. Since SVP works with families of threads that resemble single-dimentional loops, transforming any kind of loop into a meaningful construct for the SVP is an important step. For that reason, loops are divided into single-dimensional and multi-dimensional ones with each category requiring a different transformation method. When it comes to single-dimensional loops, a combination of parallel executing independent dataflows (data-chains) and use of SVP’s synchronization primitives to reduce accesses to the main global memory brings tremendous increases in speedup and efficiency.

Multi-dimensional loops that contain static loop carried dependences are of main concern. Lamport’s hyperplane method is applied; however, with a twist: Instead of precomputing any loop transformation, it is up to the run-time environment to intuitively follow the dependence vector over the index space and discover the different hyperplanes per cycle. This novel idea gave birth to our first run-time algorithm: The fixed-size algorithm. It has the ability to apply the hyperplane idea, discovered while running the actual computation code, into the various tiles of a fixed size which divide the innermost dimension of the loop. The fixed size algorithm proved to work properly, however for optimal or even good results the size of the tile was needed to be known beforehand, effectively making the whole algorithm not particularly useful except as a stepping stone and also a great tool for comparisons. This glaring weakness of the Fixed-Size algorithm was covered by its evolutionary “descendant”: the Self-Adaptive algorithm. Working on the same principles as the Fixed-Size one, it can, at run-time, determine the optimal tile size to use at any given computation cycle by reducing it or increasing it according to the current needs.

Experimental results indicate that not only the Self-Adaptive algorithm fares very well when compared with the Fixed-Size one’s optimal results, it is also shown that for that particular type of parallelism (run-time execution of parallel families discovered on the spot) the results obtained are the best possible results that can be obtained. The algorithms are also compared with a standard compile-time method (the hyperplane method). The numbers differ, however the versatility offered by a run-time system (like dealing with irregular index spaces) makes the Self-Adaptive algorithm especially appealing.