Abstractul tezei de doctorat a lui Alexandru Emilian Şuşu

În aceste zile experimentăm un moment fertil pentru calculul paralel, în special datorat avansului în tehnologia compilării, maturarea formalismelor matematice pentru aceasta şi popularizarea proiectării hardware.

Compilarea secvențială a programelor C este dificilă pentru acceleratorul competitiv vectorial larg Connex-S care este scalabil şi customizabil, pentru aplicații îmbarcate cu 32 până la 4096 unități de execuție (*lanes*) cu 16 biți întregi şi o capacitate limitată locală de memorie de tip *scratchpad*.

Compilatorul nostru foloseşte frameworkul LLVM şi generează cod pentru OPINCAA, un asamblor vectorial de tip JIT (*Just-In Time*) şi bibliotecă C++ de coordonare pentru acceleratorul Connex-S de pe lângă un CPU. De aceea, adresăm în middle-end-ul compilatorului aspecte de vectorizare, comunicare şi sincronizare eficientă. Efectuăm analiză statică cantitativă a programului de compilat folositoare printre altele pentru alocatorul de memorie de mărime simbolică şi mecanismului de coordonare al lui OPINCAA. De asemenea, prezentăm back-end-ul LLVM pentru procesorul Connex-S şi metodologia de generare automată a codului de selectare a instrucțiunilor pentru emularea eficientă a operatorilor aritmetici şi logici pentru tipuri non-native cum ar fi întregi pe 32 bit şi floating point pe 16 biți.

Prin folosirea asamblorului vectorial de tip JIT şi prin introducerea lungimii vectoriale a lui Connex-S ca un parametru în programul OPINCAA generat, obținem portabilitate în funcție de lungimea vectorială pentru a suporta execuția pe dispozitive îmbarcate distincte, cum ar fi o serie de camere digitale cu rezoluții diferite, fiecare echipată cu acceleratoare Connex-S de lungime vectorială ajustată (*custom-width*) menită să salveze energia pentru kerneluri de procesare de imagine.

Deoarece Connex-S are o memorie locală de tip *scratchpad* de capacitate limitată de 256 KB în mod normal, noi prezentăm cum folosim de asemenea generatorul PPCG de cod de la C la C, să efectuăm *data tiling* pentru a minimiza timpul total de execuție al kernelului, astfel încât să încapă datele de intrare ale programului mai mari segmentate în memoria locală. Pentru aceasta am construit un model de cost precis pentru acceleratorul Connex-S pentru a alege la compilare mărimea de tilizare optimă.

Compilăm cu succes câteva benchmark-uri frecvent folosite, spre exemplu, în aplicații îmbarcate de înaltă performanță şi în de analiză de imagine. Raportăm îmbunătățiri de performanță de până la 11.33 când le rulăm pe acceleratorul Connex-S cu 128 de unități de execuție întregi pe 16 biți față de procesorul gazdă ARM Cortex A9 dual-core care rulează la o frecvență de 6.67 ori mai mari cu două unități SIMD Neon de 128-bit.

La sfârşitul acestei teze, propunem o metodă simplă matematică pentru a specifica toate transformările utile de analiză de imagine disponibile spre exemplu în bine-cunoscuta bibliotecă OpenCV. Folosim un *Design Specification Language* similar cu bine-cunoscutul limbaj Halide şi prezentăm un nou mod de implementare eficientă a transformărilor de analiză de imagine a unei aplicații complexe moderne de analiză cu matrice rare. Codul rezultat foloseşte biblioteca secvențială CSparse şi atinge o îmbunătățire medie a performanței de 8.32 ori pe un core x86 de 2.66 GHz. De asemenea, scriem manual cod de asamblare implementând transformări de analiză de imagine cu matrice rare pentru procesorul vectorial Connex-S.

<u>În engleză:</u>

These days we experience a very fertile moment for parallel computing, mostly due to advances in compiler technology, the maturation of mathematical formalisms for it and the popularization of hardware design.

Compiling sequential C programs for Connex-S, a competitive, scalable and customizable, wide vector accelerator for intensive embedded applications with 32 to 4096 16-bit integer lanes and a limited capacity local scratchpad memory, is challenging.

Our compiler toolchain uses the LLVM framework and targets OPINCAA, a JIT vector assembler and coordination C++ library for Connex-S accelerating computations for an arbitrary CPU. Therefore, we address in the compiler middle end aspects of efficient vectorization, communication, and synchronization. We perform quantitative static analysis of the program useful, among others, for the symbolic-size compiler memory allocator and the coordination mechanism of OPINCAA. We also discuss the LLVM back end for the Connex-S processor and the methodology to automatically generate instruction selection code for emulating efficiently arithmetic and logical operations for non-native types such as 32-bit integer and 16-bit floating-point.

By using JIT vector assembling and by encoding the vector length of Connex-S as a parameter in the generated OPINCAA program, we achieve vector-length agnosticism to support execution on distinct embedded devices, such as several digital cameras with different resolutions, each equipped with custom-width Connex-S accelerators meant to save energy for the image processing kernels.

Since Connex-S has a limited capacity local scratchpad memory of 256 KB normally, we present how we also use the PPCG C-to-C code generator to perform data tiling to minimize the total kernel execution time, subject to fitting larger program data in the local memory. We devise an accurate cost model for the Connex-S accelerator to choose optimal performance tile sizes at compile time.

We successfully compile several simple benchmarks frequently used, for example, in high performance and computer vision embedded applications. We report speedup factors of up to 11.33 when running them on a Connex-S accelerator with 128 16-bit integer lanes w.r.t. the dual-core ARM Cortex A9 host clocked at a frequency 6.67 times higher, with a total of two 128-bit Neon SIMD units.

At the end of the thesis, we also propose a simple mathematical way to specify all useful computer vision transformations available for example in the well-known OpenCV library. We employ a Design Specification Language similar to the well-known Halide and present a novel way to implement efficiently computer vision transformations of a complex modern vision pipeline with sparse matrices. The resulting code uses the sequential CSparse library and achieves an average performance improvement of 8.32 times on one x86 CPU core at 2.66 GHz. We also write manually assembler code implementing such computer vision transformations with sparse matrices for our Connex-S vector processor.