Politecnico di Milano - University of Illinois Chicago
HPPS Project

QCADESIGNER POWERED BY CUDA

Instructor: Prof.ssa Sciuto Donatella
Tutor: Santambrogio Marco Domenico

Authors:
Gibilisco Giovanni Paolo, Matr. 755066
Marconi Francesco, Matr. 755439
Miglierina Marco, Matr. 754848

2009-2010

Contents

List of Figures

Abstract

2 righe sul lavoro, -¿VENDITI BENE

State of the Art - QCA * QCADesigner * Two Engines: BISTABLE and COHERENCE, describe them shortly (see MINA site) -¿ BISTABLE IS JUST A FAST APPROXIMATION to test circuits - CUDA

Rationale

- Why QCA? * novel emerging paradigm * 2 Thz, low energy consumption and miniaturization * quantum computing - QCADesigner simulator slow on big circuits: * Every sample, each cell's polarization is computed based on the values of his neighbors, sequentially. * Bottleneck from profiling (table with times) * Simulation core: pseudo code * Identical operations repeated for each cell, big circuits -¿ thousands of cells * We chose to speedup this part of the code with CUDA because: SIMD architecture (SIMT): single instruction repeated on different data hundreds of core -¿ many threads running simultaneously, each thread responsible of computing a cell's polarization scalable, adding new cores implies a greater number of cells computed simultaenously, higher speedup - Objective * Speed up simulation for big circuits * batch simulator * Given a file .qca -¿ produce output: binary, continous values, plot on png, log with info of simulation * if same .qca -¿ same results CPU and CUDA implementation

- First approach * Downloaded from MINA the latest version of QCADesigner, NOT COMPILES! * At first, lot of work done to obtain a working batch simulator on CPU * Meanwhile analisys of the code, location of possible bottlenecks, analisys of data structures and their possible transformation in order to obtain best explotation of CUDA * Batch simulator on CPU ready -¿ start profiling (table) and location of bottleneck. - CUDA implementation * CPU algorithm -¿ CUDA algorithm proposed (working on old values each iteration, with Konrad's blessing): pseudo code. * After first implementation: wrong results, cells' polarizations don't converge, oscillations. -¿ Bistable approximations doesn't work with this algorithm. -¿ Solution: Don't change CPU algorithm -¿ don't compute neighbour cells values simultaneously. -¿ parallelization? Coloring algorithm -¿ randomization? randomize color order each sample * Cuda main data structures: arrays of polarizations, neighborhood, clocks, kink energies, stability. * Memory occupation on global, coalescent accesses, only one vector for old and new polarization * Constant memory, variables * shared memory for arrays of indexes of inputs and outputs (many accesses during kernel execution) * Moved clock values and input values calculation (only when they change) inside the kernel * Memory transfers * Fast math and float for faster approximation (only one double FU per SM) * Compulsory divergence reading neighbours - Discussion pre-results on core simulation: * Iterations per sample: ca. 5-10 * Colors: ca. 15-20 * Number of samples: 2000*2$^{n}umber_of_inputs * OnCPUcomplexity : O(samplesxmeaniterationspersamplexcells)$ $O(2000* 2^N inputx10xNcells) = O(C*2^N) * Max(theoretical)speedupachievable = cells/colors- > O(samplexmeaniterationspersamplexcolors)$ $O(2000*2^N inputsx10x15) = O(2^N)- > Technologyconstraints : *Memorytransfers : eachiteration : device- > host, stability : cellsx1byteeachsample : device- > host, outputs : number_of_outputsx8bytesmain_kernelcalls : samplesxmeaniterationspersamplexcolorsupdateinp$
$2x2^numbe r_of_inputs(onceevery1000samples)mainkernel : -1globalcoalescent- > shared : number_of_outputsx8bytes(output_indexes)-3xglobalreadcoalescent : 3xnumber_of_cellsx4bytes(neighbours, cells_colors, cells_clock)---forcycle : (2coalescent + 1random)xmaxnumberofneighbours- > bigdivergency - 1gloabalread+2globalwritecoalescent : number_of_cellsx(8+1+8)bytes(polarization, stability, pola$
$number_of_outputsreadsinshared-writeonglobalnoncoalescenttransferrate?$

Conclusions

References