Thesis projects for CS4490

Marc Moreno Maza

Ontario Research Center for Computer Algebra (ORCCA)
University of Western Ontario, Canada

September 19, 2016

Research themes and team members

- Symbolic computation: computing exact solutions of algebraic problems on computers with applications to sciences and engineering.
- High-performance computing: making best use of modern computer architectures, in particular hardware accelerators (multi-cores GPUs)

Current students

PhD: Parisa Alvandi, Ning Xie, Mahsa Kazemi, Ruijuan Jing, Xiaohui Chen, Steven Thornton, Robert Moir, Egor Chesakov

MSc: Masoud Ataei, Yiming Guan, Davood Mohajerani

Alumni

Moshin Ali (ANU , Australia) Jinlong Cai (Microsoft , USA), Changbo Chen (Chinese Acad. of Sc.), Svyatoslav Covanov (U. Lorraine , France) Akpodigha Filatei (Guaranty Turnkey Systems ltd , Nigeria) Oleg Golubitsky (Google Canada) Sardar A. Haque (GeoMechanica , Canada) Zunaid Haque (IBM Canada) François Lemaire (U. Lille 1 , France) Farnam Mansouri (Microsoft , Canada) Liyun Li (Banque de Montréal , Canada) Xin Li (U. Carlos III , Spain) Wei Pan (Intel Corp. , USA) Sushek Shekar (Ciena , Canada) Paul Vrbik (U. Newcastle , Australia) Yuzhen Xie (Critical Outcome Technologies , Canada) Li Zhang (IBM Canada) . . .

Solving polynomial systems symbolically

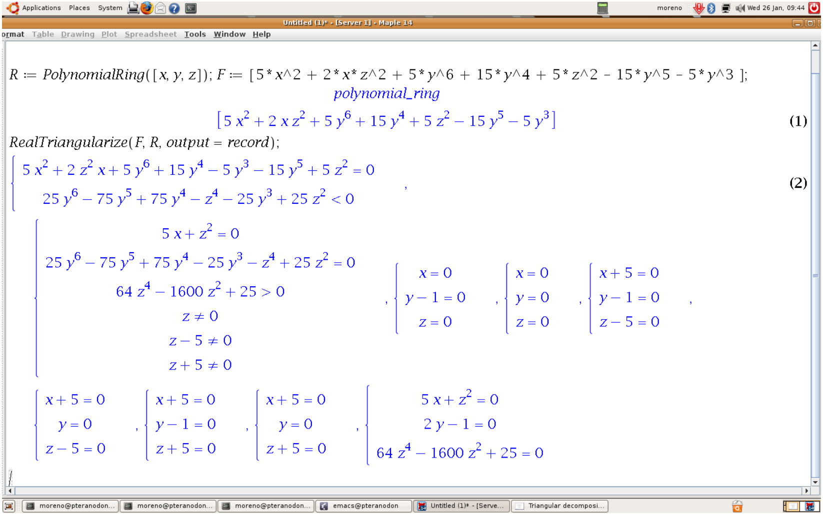
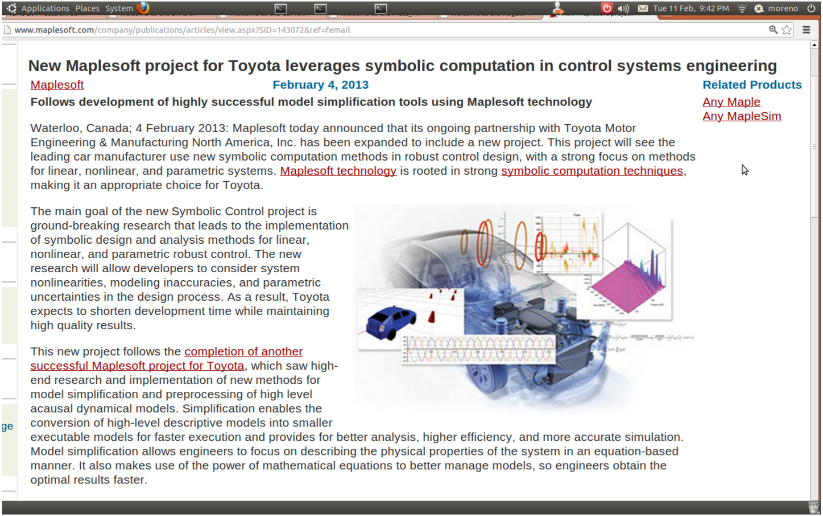


Figure: The *RegularChains* solver designed in our UWO lab is at the heart of Maple, which has about 5,000,000 licences world-wide.

Application to mathematical sciences and engineering



Applications Places System Tue 11 Feb, 9:42 PM moreno

www.maplesoft.com/company/publications/articles/view.aspx?SID=143072&ref=femail

New Maplesoft project for Toyota leverages symbolic computation in control systems engineering

Maplesoft February 4, 2013

Follows development of highly successful model simplification tools using Maplesoft technology

Waterloo, Canada; 4 February 2013: Maplesoft today announced that its ongoing partnership with Toyota Motor Engineering & Manufacturing North America, Inc. has been expanded to include a new project. This project will see the leading car manufacturer use new symbolic computation methods in robust control design, with a strong focus on methods for linear, nonlinear, and parametric systems. Maplesoft technology is rooted in strong symbolic computation techniques, making it an appropriate choice for Toyota.

The main goal of the new Symbolic Control project is ground-breaking research that leads to the implementation of symbolic design and analysis methods for linear, nonlinear, and parametric robust control. The new research will allow developers to consider system nonlinearities, modeling inaccuracies, and parametric uncertainties in the design process. As a result, Toyota expects to shorten development time while maintaining high quality results.

This new project follows the completion of another successful Maplesoft project for Toyota, which saw high-end research and implementation of new methods for model simplification and preprocessing of high level acausal dynamical models. Simplification enables the conversion of high-level descriptive models into smaller executable models for faster execution and provides for better analysis, higher efficiency, and more accurate simulation. Model simplification allows engineers to focus on describing the physical properties of the system in an equation-based manner. It also makes use of the power of mathematical equations to better manage models, so engineers obtain the optimal results faster.

Related Products

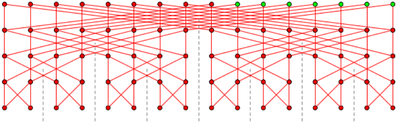
Any Maple

Any MapleSim

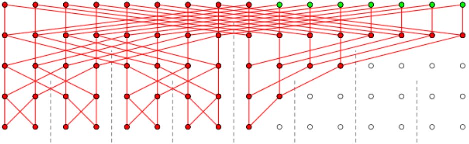
Figure: Toyota engineers use our software to design control systems

Project 1: Truncated Fourier Transform

1. The *Fast Fourier Transform* (FFT) is a kernel in scientific computing
2. It maps a vector of size 2^e to another vector of size 2^e
3. The *Truncated Fourier Transform* (TFT) supports arbitrary vectors but is challenging to implement, in particular on multi/many-cores



FFT with *artificial* zero points

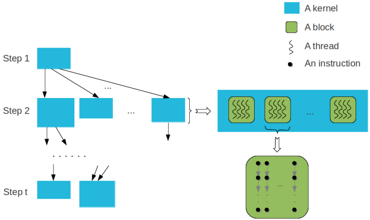


TFT removes unnecessary computations

Objectives

1. Realize an implementation of the TFT and its inverse map
2. A configurable `Python` script will generate the `CilkPlus` code within the BPAS library `www.bpaslib.org`

High-performance computing: models of computation



Let $\mathbb{K}$ be the maximum number of thread blocks along an anti-chain of the thread-block DAG representing the program $\mathcal{P}$. Then the running time $T_{\mathcal{P}}$ of the program $\mathcal{P}$ satisfies:

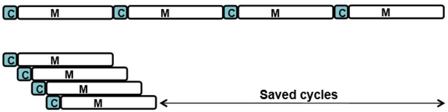
$$T_{\mathcal{P}} \leq (N(\mathcal{P})/\mathbb{K} + L(\mathcal{P}))\, C(\mathcal{P}),$$

where $C(\mathcal{P})$ is the maximum running time of local operations by a thread among all the thread-blocks, $N(\mathcal{P})$ is the number of thread-blocks and $L(\mathcal{P})$ is the span of $\mathcal{P}$.

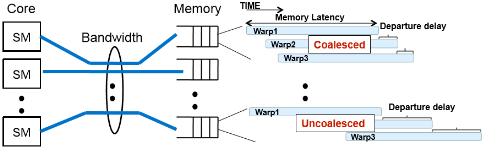
Our UWO lab develops mathematical models to make efficient use of hardware acceleration technology, such as GPUs and multi-core processors. This project is supported by IBM Canada.

Project 2: Models of computation for GPUs

1. Several models of computations attempt to estimate the performance of algorithms (or programs) targeting GPGPUs
2. The MWP-CWP Model analyzes how computations and memory accesses are interleaved in GPU programs
3. The MCM focuses on memory access patterns and memory traffic in GPU algorithms



MWP-CWP Model



MCM Model

Objectives

1. Compare those models on well-known kernels of scientific computing
2. Can we unify then?

High-performance computing: parallel program translation

```
int main(){
  int sum_a=0, sum_b=0;
  int a[ 5 ] = {0,1,2,3,4};
  int b[ 5 ] = {0,1,2,3,4};
  #pragma omp parallel
  {
    #pragma omp sections
    {
      #pragma omp section
      {
        for(int i=0; i<5; i++)
          sum_a += a[ i ];
      }
      #pragma omp section
      {
        for(int i=0; i<5; i++)
           sum_b += b[ i ];
      }  }  }
}
```

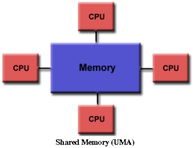
```
int main()
{
  int sum_a=0, sum_b=0;
  int a[ 5 ] = {0,1,2,3,4};
  int b[ 5 ] = {0,1,2,3,4};

  meta_fork shared(sum_a){
    for(int i=0; i<5; i++)
       sum_a += a[ i ];
      }

  meta_fork shared(sum_b){
    for(int i=0; i<5; i++)
       sum_b += b[ i ];
      }

  meta_join;
}
```

```
void fork_func0(int* sum_a,int* a)
{
        for(int i=0; i<5; i++)
          (*sum_a) += a[ i ];
}
void fork_func1(int* sum_b,int* b)
{
        for(int i=0; i<5; i++)
           (*sum_b) += b[ i ];
}
int main()
{
  int sum_a=0, sum_b=0;
  int a[ 5 ] = {0,1,2,3,4};
  int b[ 5 ] = {0,1,2,3,4};
  cilk_spawn fork_func0(&sum_a,a);
  cilk_spawn fork_func1(&sum_b,b);
  cilk_sync;
}
```

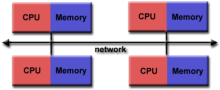
Our lab develops a compilation platform for translating parallel programs from one language to another; above we translate from `OpenMP` to `CilkPlus` through `MetaFork`. This project is supported by IBM Canada.

Project 3: Integrating NPI support into METAFORK

1. Currently, the METAFORK language supports different schemes of parallelism: fork-join, pipelining, Single-Instruction Multi-Data.
2. CILKPLUS, OPENMP, CUDA code can be generated from METAFORK code by the METAFORK compilation framework



Shared memory



Non-shared memory

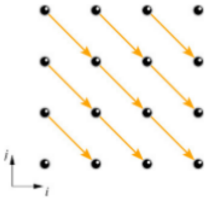
Objectives

1. Enhance the METAFORK language and METAFORK compilation framework to support non-shared memory and generate MPI code.
2. This linguistic extension should be compact while allowing to generate efficient MPI code.

High-performance computing: automatic parallelization

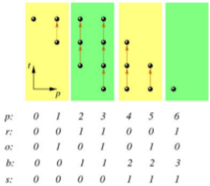
Serial dense univariate polynomial multiplication

```
for(i=0; i<=n; i++){
  for(j=0; j<=n; j++)
    c[i+j] += a[i] * b[j];
}
```



GPU-like multi-threaded dense univariate polynomial multiplication

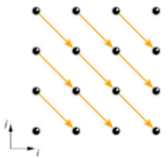
```
meta_for (b=0; b<= 2 n / B; b++) {
    for (u=0; u<=min(B-1, 2*n - B * b); u++) {
       p = b * B + u;
       for (t=max(0,n-p); t<=min(n,2*n-p) ;t++)
           c[p] = c[p] + a[t+p-n] * b[n-t];
   }
}
```



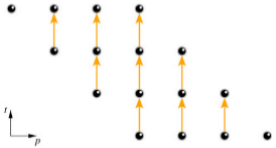
We use symbolic computation to automatically translate serial programs to GPU-like programs. This project is supported by IBM Canada.

Project 4: Dependence analysis for parametric GPU kernels

1. For performance and portability reasons, GPU kernels should depend on program and machine parameters.
2. Standard software tools for automatic parallelization do not support parametric GPU kernels. But MetaFork almost does . . .



Input iteration space



Iteration space after change of coordinates

Objectives

1. Extend the MetaFork framework with a software component for doing dependence analysis on parametric code.
2. Note that the MetaFork framework already has the infrastructure to generate parametric GPU kernels.

Research projects with publicly available software

Basic Polynomial Algebra Subprograms

www.bpaslib.org

www.metafork.org

www.cumodp.org

www.regularchains.org