EE482C Project Proposal
Brook Compilation

Jayanth Gummaraju, Ahbishek Das, and Mattan Erez

Tuesday, 7 May 2002

Abstract

In this project we will develop a basic compiler for Brook. Our aim is to provide the infrastructure for compiling Brook down from the meta-compilation level to the hardware. Our initial hardware model is Imagine macro-code and KernelC.

1 Introduction

Brook [Buc02] is a high-level streaming dialect of C. It provides constructs for defining streams and kernels, and syntax for applying kernels to streams. In addition, a limited set of stream operators is defined for manipulating streams as multi-dimensional arrays and stream derivations. The combination of these mechanisms defines a stream program that exposes parallelism and locality without hardware specific annotations. The task of the Brook compiler is to compile a Brook program down to a streaming hardware implementation. This will be done in three major steps:

- Front-end parsing
- Compilation into a Streaming Virtual Machine
- Code generation for specific hardware configuration

In this project we will concentrate on the second step of compiling Brook to the SVM level. We will ignore kernel compilation and multi-node issues for now. Also, due to the fact that the SVM is not yet well-defined we will target Imagine macro-code [MK98] and KernelC [MKOR01] instead.

2 Goals

This project has three main goals:

1. Define and develop a basic infrastructure for compilation
2. Demonstrate and test the framework by compiling StreamMD
3. Provide simple optimization passes

2.1 Basic Compilation Infrastructure

The frontend processing of Brook is performed by meta-compilation [Che02]. The compilation framework will interface with the meta-compiler and process its output into a suitable *intermediate representation* which we will define.

After expressing the program in the IR we will perform very limited analysis in order to compile it to Imagine macro-code. We will print the kernels as is and manually re-write them in KernelC initially.

The output of this compilation stage is a macro-code program that has the following features:

- Memory is allocated for all streams.
- A control-flow graph is generated and strictly adhered to (no re-ordering of operations).
- All stream operations (kernels and memory) run to completion before the next one is started.
- All stream operators such as self-product are expanded by in-lining appropriate code and/or kernels.
- Kernel execution is stripped such that for every strip, all the inputs and outputs fit into the SRF.
- Each kernel strip is processed independently with no double buffering.
- Scalar reductions are identified and marked. For stripped kernels, the reduction variable is recirculated into the kernel to accomplish reduction across the strips.
- Stream reductions are identified and handled. An index is associated with every partial update, the stream of partials is then sorted by this index and finally reduced.

2.2 Evaluation

We will evaluate the basic framework by compiling the non-gridded version StreamMD A. Initially kernels will be manually re-written in KernelC, and the rest of the program will be automatically compiled. We will test the resulting Imagine program on ISIM.

2.3 Optimizations

Once we verify the basic compilation process we will begin implementing simple optimizations. For this project we plan to support double-buffering and simple dependence analysis to allow straightforward strip-mining (running multiple kernels on a single strip of data).

Once an optimization is implemented we will evaluate its benefit by comparing the run-time and bandwidth requirements of the optimized program against the original code produced.

3 Schedule

- 16 May – Interface with the meta-compiler and define an IR
- 28 May – Perform analyses and queries on the IR (simple kernels, reductions, stream operators, record sizes ...)
- 31 May – Basic compilation complete
- 4 June – Evaluation of simple compilation and coding of optimizations
- 6 June – Report

References

[Buc02] Ian Buck, *Brook language specification*, 2002, SSS Internal Document.

[Che02] Ben Chelf, *Brook metacompilation*, SSS Internal Document, 2002.

[MK98] Peter Mattson and Ujval Kapasi, *Imagine macrocode*, Imagine internal document, 1998.

[MKOR01] Peter Mattson, Ujval Kapasi, John Owens, and Scott Rixner, *Imagine programming system user's guide*, Imagine internal document, 2001.

A StreamMD Brook Code

The following pages contain the StreamMD code with many kernel computations omitted. The main loop and the important record and kernel definitions are printed below.

```
struct waterMolecule_struct {
  vec3 o;
  vec3 h1;
  vec3 h2;
  double doh1eq;
  double doh2eq;
  double dh1h2eq;

};

typedef stream struct waterMolecule_struct * waterMolecule;
typedef stream waterMolecule (* waterMoleculePair) [2];

/* a molclField stream will store forces or velocities
 * for each atom in a molecule
 */

struct molclField_struct {
  vec3 o;
  vec3 h1;
  vec3 h2;
};

typedef stream struct molclField_struct * molclField;
typedef stream struct molclField_struct * (* molclFieldPair) [2];

kernel void MolclInteractions (waterMoleculePair pstn,
          reduce molclFieldPair force,
       reduce double * wnrg);

kernel void MolclSpringForces ( waterMolecule pstn,
reduce molclField force,
reduce double * sprnrg);

kernel void VelocUpdate ( molclField force,
reduce molclField veloc);

kernel void PostnUpdate ( molclField veloc,
reduce waterMolecule pstn);

kernel void KineticEnergy (molclField veloc, reduce double * kinnrg);


main() {
  waterMolecule pos0;
  waterMolecule pos1;
  waterMolecule pos3;
  rms pos_accuracy;
  molclField force;

  /* Data common to both run */

  MolclSetLength (coord_file, pos3);
```

```
  /* Init force stream */
  FieldLoad (force_file,force);

  /* Time step */
  POW2 = 2;
  DELTAT = 1e-2/(1<<POW2);

  /* Total simulation time */
  TOTAL_TIME = 100.000001 * DELTAT;

  /* Max number of time steps */
  MAXSTEPS = int(TOTAL_TIME/DELTAT);

  {
    int i;
    FILE * f_energy;
    double nrg0,nrg_variance;

    double wnrg = 0.0, sprnrg = 0.0, kinnrg = 0.0, totnrg = 0.0;

    molclField veloc;

    /* Init position with pos3 */
    MolclLoad (coord_file, pos0);
    copy(pos3,pos0);

    /* Init velocity */
    FieldLoad (velocity_file,veloc);

    f_energy  = fopen("Energy","w");
    f_position = fopen("Position","w");

    printf("\n");
    printf("Number of steps for this simulation:\t%d\n",MAXSTEPS);
    printf("Time step:\t%e\n",DELTAT);
    printf("Total simulation time:\t%e\n",DELTAT*MAXSTEPS);

    /* Value for the first step */
    nrg_variance = -1.;

    /* Main time iteration loop */
    for (i=0;i<MAXSTEPS;++i) {

waterMoleculePair pospairs;
molclFieldPair forcepairs;

if (every(i,MAXSTEPS,10)) {
printf("Time step is %d ; ",1+i);
printf("\tVariance of energy (dt^2) = %e\n",sqrt(nrg_variance/i)/fabs(nrg0));
}

/* zero forces */
ZeroField(force);
```

```
/* printf ("\nDoing Interaction...\n\n"); */

/* Computation of forces */

/* Compute long ranged forces */
wnrg=0;

pospairs = pos0.selfproduct();
forcepairs = force.selfproduct();

MolclInteractions (pospairs, forcepairs, &wnrg);

/* Compute bond forces at time t */
sprnrg=0;
MolclSpringForces (pos0, force, &sprnrg);

/* Computation of forces is now complete. */

/* update velocity from t-dt/2 to t */
/* v(t) = v(t-dt/2) + dt/2 * Force(t) */
VelocUpdate (force, veloc);

/* Check for energy conservation */

kinnrg=0;
KineticEnergy(veloc,&kinnrg);

/* Total energy */
totnrg = wnrg + sprnrg + kinnrg;

/* Variance of energy */
if (nrg_variance == -1.) {
nrg0 = totnrg;
nrg_variance = 0.;
}
else {
nrg_variance += (totnrg-nrg0)*(totnrg-nrg0);
}

/* update velocity from t to t+dt/2 */
/* v(t+dt) = v(t) + dt/2 * Force(t) */
VelocUpdate (force, veloc);

/* Update position using v(t+dt/2) */
/* pos(t+dt) = pos(t) + dt * v(t+dt/2) */
PostnUpdate (veloc, pos0);

if ( every(i,MAXSTEPS,100) ) {
/* Write energy statistics */
fprintf(f_energy,"%d %e %e %e %e\n",i,totnrg,wnrg,sprnrg,kinnrg);

/* write position of water molecules to file */
PosPrint(pos0, "");
}
```

```
}

    nrg_variance = sqrt(nrg_variance/(MAXSTEPS-1));
    nrg_variance /= fabs(nrg0);

    printf("\nVariance of Energy (dt^2) = %e\n",nrg_variance);

    fclose(f_energy);
    fclose(f_position);
  }
}
```