Autogen x Legion

Jesse Wang - Under the guidance of Professor Pramod Ganapathi and Mohammad Javanmard

An Introduction to Autogen

- Autogen is an algorithm developed by our own professors at Stony Brook University that takes in dynamic programming problems and discovers “highly efficient cache-oblivious parallel recursive divide-and-conquer algorithms from inefficient descriptions of DP recurrences”. The paper that describes the Autogen algorithm is cited below.

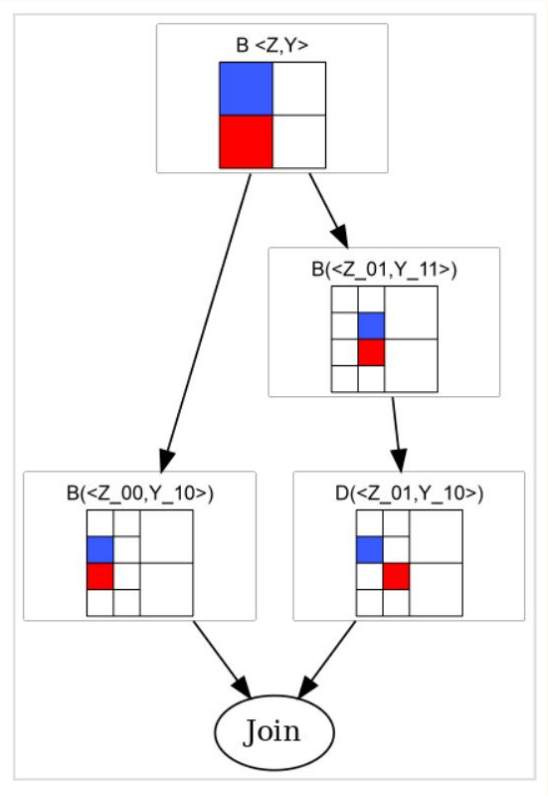
Rezaul Chowdhury, Pramod Ganapathi, Stephen Tschudi, Jesmin Jahan Tithi, Charles Bachmeier, Charles E. Leiserson, Armando Solar-Lezama, Bradley Kuszmaul, and Yuan Tang. Autogen: Automatic Discovery of Efficient Divide-&-Conquer Algorithms for Solving Dynamic Programming Problems. Transactions on Parallel Computing (TOPC). 2017. Volume 4. Issue 1. Article 4. Pages 1-30.

Online link: https://dl.acm.org/doi/pdf/10.1145/3125632

An (Easier) Introduction to Autogen

Naive Implementation of dynamic programming problem

Magic!



highly efficient cache-oblivious parallel recursive divide-and-conquer algorithms in the form of a directed acyclic graph!

Recent developments...

Advancement of Modern Computer Architectures

Increase in heterogeneous (multi-processors/cores) machines and deep memory hierarchies

High Performance + Power Efficiency

Cost of data movement is starting to dominate cost of computation in both power and performance

An Introduction to Legion

- Legion is a data-centric parallel programming system for writing portable high-performance programs targeted at distributed heterogeneous architectures.

- Legion has the ability to automate many tedious tasks by correctly extracting task and data-level parallelism and moving data around complex memory hierarchies.

https://legion.stanford.edu/

Benefits of Legion

User-Specification of Data Properties

Legion gives abstractions for programmers to declare properties of program data.

Automated Mechanisms

Legion can implicitly extract parallelism and issue the necessary data movement operations in accordance with the application-specified data properties.

User-Controlled Mapping

Legion enables easy porting of applications to new architectures.

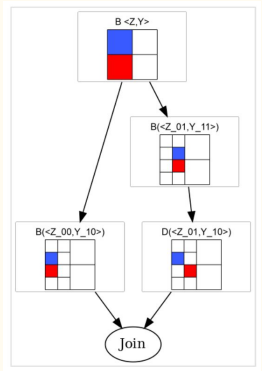
Example of Legion Code (681 lines)

```
void a_legion_task(const Task *task, const std::vector<PhysicalRegion> &regions, Context ctx, HighLevelRuntime *runtime){
    Argument args = task->is_index_space ? *(const Argument *) task->local_args
    : *(const Argument *) task->args;
    LogicalRegion lr = regions[0].get_logical_region();
    int size = args.size;
    int half_size = size/2;
    cout<< "A" << args.size << " " << args.top_x1 << " " << args.top_y1 << " " << args.top_x2 << " "<< args.top_y2 << " \n";
    if(size <= legion_threshold) {
        cout<< "SERIAL\n";
        TaskLauncher A_Serial(A_NON_LEGION_TASK_ID, TaskArgument(&args,sizeof(Argument)));
        A_Serial.add_region_requirement(RegionRequirement(lr,READ_WRITE,EXCLUSIVE,lr));
        A_Serial.add_field(0,FID_X);
        runtime->execute_task(ctx,A_Serial);
    }
    else{
        int tx = args.top_x1;
        int ty = args.top_y1;
        DomainPointColoring coloring;
        IndexSpace is = lr.get_index_space();
        int add = half_size-1;
        LogicalPartition lp;
        if(!runtime->has_index_partition(ctx,is,args.partition_color)){
            coloring[0] = Domain(Rect<2>(make_point(tx, ty), make_point(tx+add, ty+add)));
            coloring[1] = Domain(Rect<2>(make_point(tx, ty+half_size), make_point(tx+add, ty+half_size+add)));
            coloring[2] = Domain(Rect<2>(make_point(tx+half_size, ty), make_point(tx+half_size+add, ty+add)));
            coloring[3] = Domain(Rect<2>(make_point(tx+half_size, ty+half_size), make_point(tx+half_size+add, ty+half_size+add)));
            Rect<1>color_space = Rect<1>(0,3);
            IndexPartition ip = runtime->create_index_partition(ctx, is, color_space, coloring, DISJOINT_KIND, args.partition_color);
            lp = runtime->get_logical_partition(ctx, lr, ip);
        }
```

Target:

automatic execution of dynamic programs efficiently on distributed-memory machines.

Autogen DAG



Autogen x Legion

Legion

UPC

Other parallel paradigm frameworks