# Applications of Parallel Computers

## Problem Description

CS267 Assignment 3: Parallelizing Knapsack

# Assignment 3: Parallelizing Knapsack

## Due Date: To be determined by your instructor

## Problem statement

The purpose of this assignment is introduction to programming in Partitioned Global Address Space (PGAS) languages. You need to parallelize a solution of the 0-1 knapsack problem using UPC on XSEDE's Blacklight supercomputer.

You have Nitems with

- positive integer weights {`w[1], ..., w[N]`} and
- positive integer values {`v[1], ..., v[N]`}

The goal is to find a subset of the items such that its total weight does not exceed given bag capacity `C` and its total value is the largest possible.

We consider solving the problem using dynamic programming. Define `T[i,j]` to be the maximum value that can be attained with weight ≤ `i` using items {`1, ..., j`}. Then it satisfies the following recurrence relation:

```
T[i,j] = max( T[i,j-1], T[i-w[j],j-1]+v[j] ).
```

The value of the optimal solution is found at `T[C,N]`. Backtracking over the choices gives the items used in the solution.

## Correctness and Performance

The code provided checks the total weight and value of the parallel solution versus the serial answer and does backtracking through the choices to figure out the items and ensure they sum up to the weight and value given.

You should note that the time for the backtracking **IS** included in the timing of your code so you should make sure both building the table and doing the backtracking are running efficiently.

The supplied (naive) parallel code uses a cyclic layout and computes entries corresponding to the same item (`T[:,j]`) in parallel. Barrier synchronization is used when proceeding to next item. This means a lot of fine-grained communication - the parallel code may run slower than the serial code!

Your goal is to get better scaling vs. the serial code. To improve parallel efficiency, consider blocking rows and/or columns (of T), and using bulk communication and/or pipelined computation. Using UPC is a requirement.

## Grading

Your grade will depend on the performance and efficiency sustained by your code on Blacklight for 2 types of scaling runs - each contributing 50% to the grade:

- small scaling runs from 1 to 8 processors
- large scaling runs from 16 up to 256 processors

For this assignment we will consider only the strong scaling efficiency. The grade is given according to the following rules:

- If the efficiency is between 0 and 0.5 you will receive a score between 0 and 75 proportional to it (Ex:0.2 gives a score of 30)
- If the efficiency is between 0.5 and 0.8 you will receive a score between 75 and 100 proportional to it (Ex:0.62 gives a score of 85)
- If the efficiency is above 0.8 you will receive a score of 100

## Source files

The starting files with their descriptions can be found in the Starting Code folder

To download all the files directly to Blacklight you can use the following command:

```
wget http://www.eecs.berkeley.edu/~penpornk/XSEDE2015/knapsack.tgz
```

## Login, Compilation and Job submission

The easiest way to login to XSEDE resources is via the Accounts tab in XSEDE User Portal. To reach this page login to XSEDE User Portal, navigate to MY XSEDE tab and from there select the Accounts page. All machines you have access to will have a **login** option next to them that will launch OpenSSH via a Java Applet. For this assignment we will be using the Blacklight supercomputer.

For setting up XSEDE single login and OpenSSH separately from XUP please check the instructions on doing so within the XSEDE User Portal; Please be aware that for direct ssh you must fillout the form present in System access for Blacklight

You can extract the downloaded directory from the tgz archive with:

```
tar -xvzf knapsack.tgz
```

The default Makefile is built with the default intel compiler; however you must first load UPC on Blacklight with the command:

```
module load sgi-upc-devel
```

Then simply type **make** to compile the code

To submit jobs on the Blacklight system you will use the qsub interface for example:

```
qsub job-serial
```

To check the status of running jobs you can use the following command

```
qstat -u $USERNAME
```

Once you have code that gives correct output you should check it's efficiency with the autograder provided by running the command:

```
qsub auto-knapsack
```

This will run your Knapsack code for multiple processor sizes to determine strong scaling speedup and efficiency

---

## Resources

- Lecture 8 - UPC, UPC Specification, Berkeley UPC Group.
- UPC Quick Reference Card, UPC-Collective Quick Reference.
- GASNet is a high-performance network layer that supports one-sided communication in UPC.
- Other PGAS languages: X10, Titanium, Co-Array Fortran, Chapel.

Last modified: Saturday, 28 March 2015, 1:57 AM

Current course

Parallel

Participants

Badges

General

Homework 0 - Describing a Parallel Application

Programming Homework 1 - Tuning Matrix Multiply

Programming Homework 2 - Parallelizing a Particle ...

Programming Homework 3 - Parallelizing Knapsack

**Problem Description**

Starting code

HW3 Assignment submission

HW3 Discussion

My courses

## ADMINISTRATION

Course administration

My profile settings