SST Challenge
SC25 IndySCC

Introduction

You have been selected as the chief architect for your lab's newest supercomputer. You have received some options from vendors, but to ensure you are getting the best value, you have decided to run some simulations to see what will work best for your lab's applications. You will use SST to help you pick the best CPU and network configurations.

Building SST

We will use **SST 15.1** for the competition, released November 4, 2025. You will need to install both SST-Core and SST-Elements 15.1. If you used a different version to prepare for the competition, please ensure you update both repositories. Links to the install directions are below. If you installed from a git repository, you can also pull the `v15.1.0_Final` branch or checkout the `v15.1.0_Final` tag (for both repos), which is the same code as the release tarball.

- SST 15.1 Build Instructions: Link
- SST 15.1 Download Location: Link

Building the Vanadis toolchain

As part of Problem 1, you will need to build a cross-compiler to compile programs to RISC-V, so that you can compile a binary that can run in Vanadis. This part of the process has not changed for SST 15.1. To build the cross compiler, follow the instructions on this page in the section "LLVM + RISCV GNU Toolchain": link.

Problems

Problem 1 - Node (30 points)

The first thing you need to do to design your new machine is pick a CPU. CPU vendors have a number of SKUs available, targeting different market segments and applications. This means you have a large design space to choose from. You will simulate a single benchmark running on the different configurations so that you can choose the one that gives you the best performance. Your budget for a single node in your system is $9100.

Instructions

1. Go to the `node/` directory
2. Use the `Makefile` to compile `beam.c` using the RISC-V toolchain
3. View the spreadsheet `cost-model.xlsx` that explains how to calculate the cost of a design
4. Run the configuration `p1.py` through SST to simulate your designs

Here is an example of running the simulation. Pass `--help` to the configuration script for information on the arguments.

```
$ sst p1.py -- -e beam -n 16 -c slow -t no -x small -y small -s private
-z small -b slow -w 6 -m basic
```

Important: We are discussing the performance of the *simulated* system and performance, measured as `1/time`. Thus, you need to consider the simulation times for the CPU configurations, not the wallclock times. When you run SST, it reports the simulation time on a line that looks like the following:

```
Simulation is complete, simulated time:  4.98240 ms
```

Remember that performance is inversely proportional to simulation time: `1/simulated time`. The highest performance system will have the lowest simulated time.
Determine the following:

1. The CPU configuration that gives the **highest performance per dollar** under the budget.
2. The CPU configuration that gives the **highest performance** under the budget.

For each item provide a list of (in this order): the simulated time, core count, core type, smt (yes or no), L1 type, L2 type, L2 organization, L3 type, network type, number of memory channels and memory type. For example:

```
4.98240ms,16,slow,no,small,small,private,small,slow,6,basic
```

Submit your answers in a document titled `p1.txt`. Put one answer per line, in the order above.

You will receive 15 points per correct answer. Partial credit will be awarded if an answer is valid (within the cost and has an accurate simulation time) and is in the top 10% of configurations for the question. Answers in the top 10% will receive 1pt; in the top 5% will receive 3pts and in the top 1% will receive 8pts.

Problem 2 - Topology (30 points)

Now that you've chosen a CPU, you'll need to figure out the network that you want to use. You have a bduget for a system with 16,384 nodes. It would take too long to simulate entire programs running on thousands of nodes, so instead, so you will use Ember to simulate only the MPI traffic. You have talked with the computer scientists are your lab and they have designed two workloads for you, one based on a 3D Halo exchange motif and one based on a Sweep motif. You will determine an optimal configuration for each.

Workloads

The two workloads can be specified in the arguments of `dragonfly.py` as follows:

1. Halo: `--app=halo --work=60 --iter=55`
2. Sweep: `--app=sweep --iter=20`

Network Configurations

You have talked to the vendors and they have suggested 6 different dragonfly topologies. You have a choice of configuring the network to use groups of either 512 or 256 endpoints, and you have a choice of three different network bisection bandwidths, dubbed `full`, `half`, and `quarter`. These each correspond to a different number of intergroup links.

To get the system size, you will multiply the number of groups with the group size. Thus, the valid system configurations are:

1. `--group_size=512 --num_groups=32`
2. `--group_size=256 --num_groups=64`

You can specify the bandwidth with `--bw=<full,half,quarter>`. Here is an example of the full set of arguments:

```
sst dragonfly.py -- --app=halo --work=60 --iter=55 --jobs=4
--group_size=512 --num_groups=32 --bw=full
```

Instructions

Go to the `network/` directory and run the both workloads on all 6 configurations using `dragonfly.py`. For each of the six network configurations, run both workloads. Determine the configuration producing the lowest simtime for both motifs. Submit the file `p2.txt` with two lines; on each line, specify the simtime, the app name, the group size, and the bandwidth choice.

```
123.32322ms,halo,512,full
123.32322ms,sweep,512,full
```

You may receive half credit if you provide only one correct line.

Problem 3 - Scaling (20 points)

As part of procuring your new computer, you have asked the vendors to provide plans to increase the system size. If this machine is successful, you may decide to purchase a second one in the future, doubling your system size.

You should evaluate how well a potential scaled system will do on much larger jobs. You have decided to swap out the Sweep workload with an FFT workload. **Take the network configurations from Problem 2, double the number of groups**, and run the following workloads:

1. Halo: `--app=halo --work=80 --iter=20`
2. FFT: `--app=fft --work=1600 --iter=20`

Important note: Only 5 of the 6 networks can have their group count doubled. The 16k network with group_size=256, bw=quarter only has one intergroup link, and doubling the number of groups would require halving this number for a fair comparison. Thus, you will only simulate 5 of the 6 networks for this problem.

Second important note: If you don't have time to do the full FFT runs, you should still run with `--iter=20`, as it will affect the output of earlier iterations. Simply kill the job whenever you run out of time.

Instructions

Run each of the 5 valid network configurations on both inputs. Submit the entire output of each simulation in the subdirectory `p3/`. The 10 files should be named as follows:

```
SST/p3/<app>_<group_size>_<bw>.txt, e.g.
SST/p3/halo_512_full.txt
```

These simulations produce output after each iteration. You can receive partial credit for partially complete simulations. Each complete iteration is worth 1/10th of a point.

Result Submission

Submit your files to the submission server. Your submission directory should look like this:

```
SST/
  p1.txt
  p2.txt
  p3/
     halo_512_64_full.txt
     halo_512_64_half.txt
     ...
```

Scoring

Your submission will be scored out of a possible 80 points. An auto-grader will be used so please ensure your files are formatted as specified in each problem.