1 Enkidu Component-Based Discrete Event Simulator

Enkidu, a component-based simulation engine, provides a framework for simulating objects representing various physical components of the system. These components will interact by passing each other parcels, much like processes pass events in a process or event based simulator.

1.1 Enkidu Goals

- Bare-bones, No-Frills, Serial Framework
- Hybrid Model: Event-driven & Time-step (monolithic)
- Low overhead (<20 host cycles per component per clock tick)

1.2 Enkidu Structure

The framework consists of components which represent objects within the simulated architecture. These components communicate by passing parcels.

When passing a parcel, the sending component invokes a function of the Enkidu API which will specify the parcel data, the destination component, and the cycle delay before the parcels arrives. The Enkidu backend will then enqueue this parcel in a priority queue sorted by arrival time. When the parcel is scheduled to arrive, the Enkidu backend will invoke a function on the destination component, passing in the parcel data.

Each component has two important functions: preTic() and handleParcels(). These correspond to actions performed before parcels for that cycle arrived and actions to handle arriving parcels in a cycle.

Pseudocode for the serial version of Enkidu appears in figure 1.

```c
priority_queue<parcel*> parcelQ;
void runSimulation() {
        while(1) {
                foreach c (components) {
                        c->preTic();
                }
                while((parcelQ.size() > 0) && ((parcelQ.top()->arrivalTime() <= _cycle))) {
                        parcel* p = parcelQ.top();
                        p->destination()->handleParcel(p);
                        parcelQ.pop
                }
        }
}
```

Figure 1: Pseudo-code for serial Enkidu

2 Internals Documentation

2.1 Class component

```
label sec:class:component
Physical Component of the system
Type abstract
```

Include file enkiduCode/enkidu_api.h
Detailed description

This class represents a component of the system such as a CPU, PIM, network link, etc. It can send and receive parcels. Implementors should inherit from this class when creating new components.

Public methods

typedef vector<component*> componentVec Vector of components

static const unsigned long long cycle () Get the number of cycles the simulation has been running

pure virtual void finish () Post-simulation finish

Called after the simulation is finished.

pure virtual void handleParcel (parcel *p) Handle incoming parcel

Called when a parcel has been delivered to a component. The component should perform whatever actions are required to handle the parcel, including generating new parcels.

virtual void idle () Perform actions if idle this cycle

Empty by default, override this function if the component needs to perform any actions at the end of the clock cycle, after all parcels have been handled and all idle() functions have been called.

virtual void postTic () Perform actions at end of clock cycle

Whenever a parcel is sent, it is entered into the queue. Each iteration, an event is grabbed from the queue and passed to the appropriate component to process.

virtual void preTic () Perform actions at start of clock cycle

Called after all parcels are handled, but before the postTic() functions have been called.

static int& keepAlive () Accessor for the keepAlive toggle

Setting the keepAlive to high allows the simulation to continue, even if there are no pending events;

static priority_queue<parcel*, vector<parcel*>, timeFunc> parcelQ The Queue of pending events

static void run () Run the simulation until done

The simulation is declared done when there are no more pending events in the queue.

static void run (unsigned int forCycles) Run the simulation for a set period of time or until done

The simulation is declared done when there are no more pending events in the queue, unless the keepAlive has been set. For each cycle, the following order is observed:

1. foreach component: preTic()
2. foreach parcel in this cycle: parcelDestination->handleParcel(parcel)
3. foreach idle component: idle()
4. foreach component: postTic()

Presently the pre/postTic() and idle() functions are called on every component. We could add an "(en/dis)able(Pre/Post)Tic()" function to improve speed if necessary.
void sendParcel (parcel *p, component* destination, int arrivalTime)
Send a parcel
Schedual a parcel to be delivered to another component.

pure virtual void setup ()
Post-constructor
Called immediately before the simulation starts. Setup which needs to be performed after all the components have been created should be performed here.

2.2 Class parcel
A parcel to be passed between components

Type instantiable

Include file enkiduCode/enkiduParcel.h

Detailed description This class represents an event passed between components. This event could be data, a thread, or run-time meta information. An attribute map of name/value pairs is provided to give an extensible interface to hold information.

Public methods

static void deleteParcel (parcel*) Destructor function
Destroys a parcel. The component which first sends a parcel should create it unless it is passing on a parcel it has already received. A component which receives a parcel should destroy it, unless it is passing it to another component.

static parcel* newParcel () Generator function
Creates a new parcel. The component which first sends a parcel should create it unless it is passing on a parcel it has already received. A component which receives a parcel should destroy it, unless it is passing it to another component.

Protected methods

parcel () Constructor
Protected. Should never be called by user, only by parcel::newParcel

~parcel () Destructor
Protected. Should never be called by user, only by parcel::deleteParcel

2.3 struct timeFunc
Functor for sorting parcels in the priority queue

Type instantiable

Include file enkiduCode/enkidu_api.h

Detailed description This is only used internally, the end user should not concern themselves with it. Basically, it just compares two parcels based on arrival time.

Public methods

bool operator() (parcel *a, parcel*b)
Sorts by arrival time

3