Data Intensive Computing on Heterogeneous Platforms

Norm Rubin
Fellow
GPG graphics products group
AMD
HPEC 2009

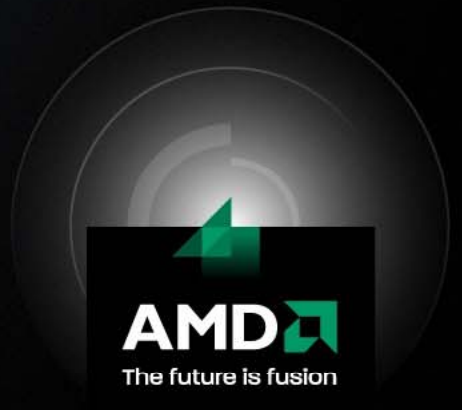
What might we see in future platforms?

Multi-core implies reprogramming so all kinds of new architectures are possible

The traditional view of more performance per year, based on clock changes is over

New approaches will change our view of computing platforms

Compute is about to change radically



The drivers for compute

1. New more natural user interfaces
2. Search (e.g., finding things in images) Massive data
3. Search over time changing data (e.g., live video) Massive data
4. Machine learning (starting with simple models) Massive data
5. Standard compute tasks

Data intensive covers 2,3, and 4

AMD
The future is fusion

Massive Data and machine learning

Data-driven models become tractable and usable

Less need for analytical models

Less need for heuristics

Real-time connectivity enables continuous model refinement

Poor model is an acceptable starting point

Classification accuracy improves over time

Simple Models

Google demonstrated the value of applying massive amounts of computation to language translation in the 2005 NIST machine translation competition.

They won all four categories of the competition translating Arabic to English and Chinese to English.

Purely statistical approach

multilingual United Nations documents comprising over 200 billion words, as well as English-language documents comprising over one trillion words.

No one in their machine translation group knew either Chinese or Arabic.

Google tops translation ranking. *News@Nature*, Nov. 6, 2006.

How Technology May Soon "Read" Your Mind

Establish the correspondence between a simple cognitive state (such as the thought of a *hammer*) and the underlying brain activity.

Use machine learning techniques to identify the neural pattern of brain activity underlying various thought processes. MRI scanner generates data

Data base of responses of individuals

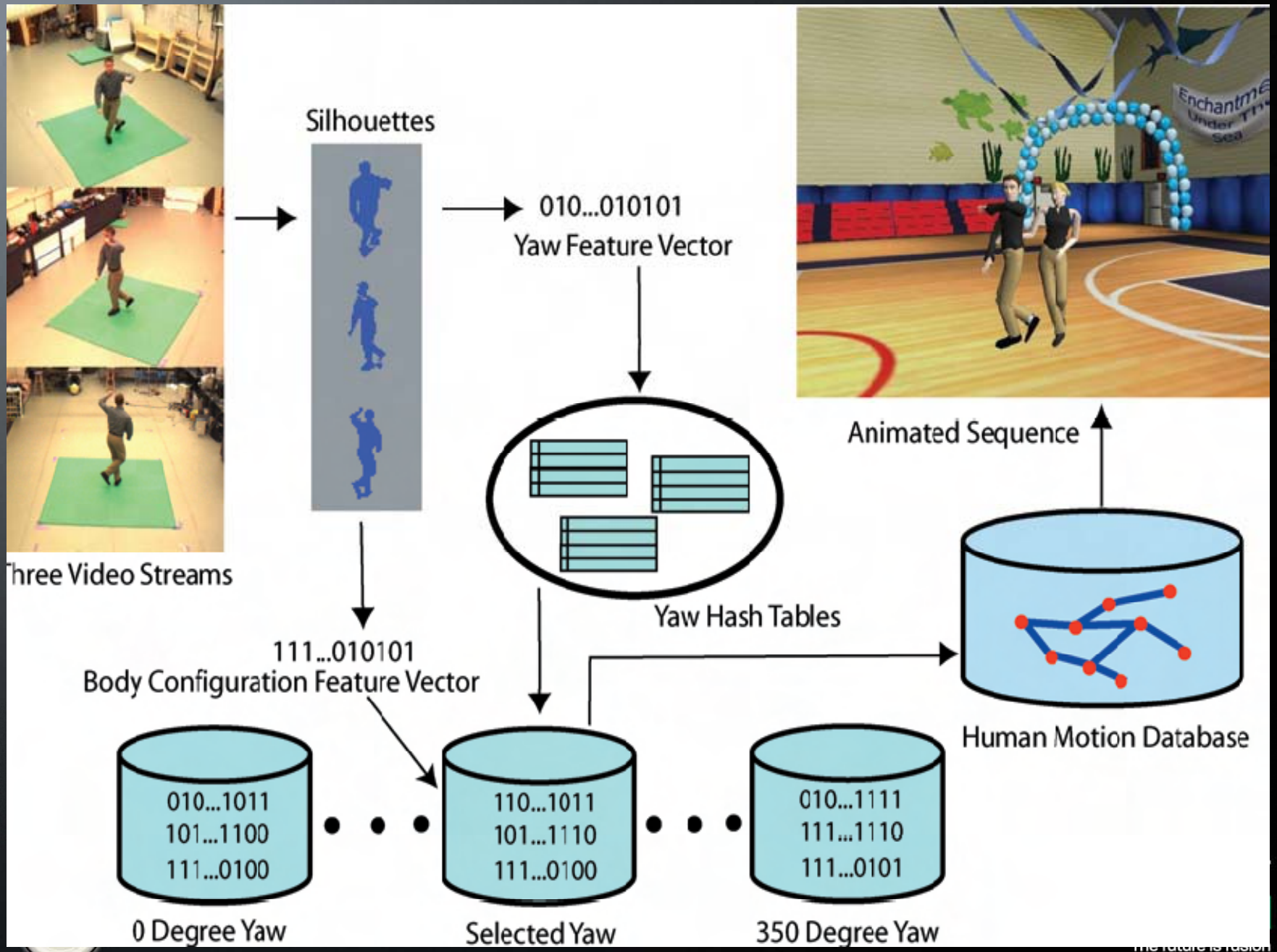
Using fMRI Brain Activation to Identify Cognitive States Associated with Perception of Tools and Dwellings
S.V. Shinkareva, R.A. Mason, V.L. Malave, W. Wang, T. M. Mitchel, and M. A. Just,
PLoS ONE 3(1): e1394. doi:10.1371/journal.pone.0001394, January 2, 2008.

New ways to program graphics

A domain expert (the programmer), 3 cameras, silhouette extraction and a large data base of human motions

Use 3D movement as a programming language

L. Ren, G. Shakhnarovich, J. K. Hodgins, H. Pfister, and P. Viola. Learning silhouette features for control of human motion. *ACM Transactions on Graphics*, 24(4), October 2005.


Silhouettes
010...010101
Yaw Feature Vector
Three Video Streams
Animated Sequence
Yaw Hash Tables
111...010101
Body Configuration Feature Vector
Human Motion Database
010...1011
101...1100
111...0100
110...1011
101...1110
111...0100
010...1111
111...1110
111...0101
0 Degree Yaw
Selected Yaw
350 Degree Yaw

How will GPU/CPU change to meet big data?

Current machine models are two separate devices

GPU – a graphics thing, maybe also a data parallel accelerator, good for lots of data, small programs

CPU – a serial processor, maybe capable of task parallelism, good for limited data, big programs

CPU strength is single thread performance

(latency machine)

GPU strength is massive thread performance

(throughput machine)

GPU compared with CPU

GPU

Great float

Great bandwidth

High performance without fixed isa, lots of room for innovation, without breaking code

different to program

But

Limited scratchpad memory in place of a cache

CPU

Great control flow

Lots of existing code

Locked into an legacy isa

Very good at reuse, real caches

But

Limited ability to scale with cores

AMD
The future is fusion

Some limitations of today's GPU

general cross thread communication

task switching, task parallel problems

nested parallelism

dynamic parallelism

very expensive to move data between GPU/CPU

programmer controlled small scratch pad memory

AMD
The future is fusion

The 3C view of the future, (c-cubed?)

We are in a time of architectural change much like the switch from mainframes to microprocessors

abundant content, connections, compute

Slash dot http://science.slashdot.org/article.pl?sid=09/08/20/1233258

NASA Probe Blasts 461 Gigabytes of Moon Data Daily

> *On its current space scouting mission, NASA's Lunar Reconnaissance Orbiter (LRO) is using a pumped up communications device to deliver 461 gigabytes of data and images per day, at a rate of up to 100 Mbps. As the first high data rate K-band transmitter to fly on a NASA spacecraft, the 13-inch-long tube, called a Traveling Wave Tube Amplifier, is making it possible for NASA scientists to receive massive amounts of images and data about the moon's surface and environment…* It kills me that the **moon** has better bandwidth than my house.

Software Challenge

Most of the successful parallel applications seem to have dedicated languages (DirectX®11/map-reduce/sawzsall) for limited domains

Small programs can build interesting applications,

Programmers are not super experts in the hardware

Programs survive machine generations

Can we replicate this success in other domains?

Can we get performance with high level languages?

CUDA and OpenCL are good languages in the hands of experts but they are still too low level

We need high level programming models that express computations over data

One possible model is “MapReduce”

Map part selects interesting data

Reduce part combines the interesting data

MapReduce

Map: for each input in parallel

- if the input is interesting – output a key and a value, values might be compound structures

Sort based on the keys

Reduce: in parallel for each key

- for each value with the same key, combine

Observations on MapReduce

Developers only write a small part of the program, rest of the code comes from libraries

No race conditions are possible

No error reporting (just keep going)

Can view the program as serial (per input)

No developer knows the number of processors

Not like pthreads

Hard part is reductions, but it appears that there are only a few that are common, so maybe they can be prebuilt

Notice nested parallelism in MapReduce

For each key (in parallel)

do a parallel reduction (also parallel)

One key may have lots of values, a second key might have a few (work stealing? Dynamic parallelism?)

K means clustering (representative algorithm)

Given N objects each with A attributes and K possible cluster centers

Map:

for each object find the distance to each center

Classify the object into a cluster, based on min distance

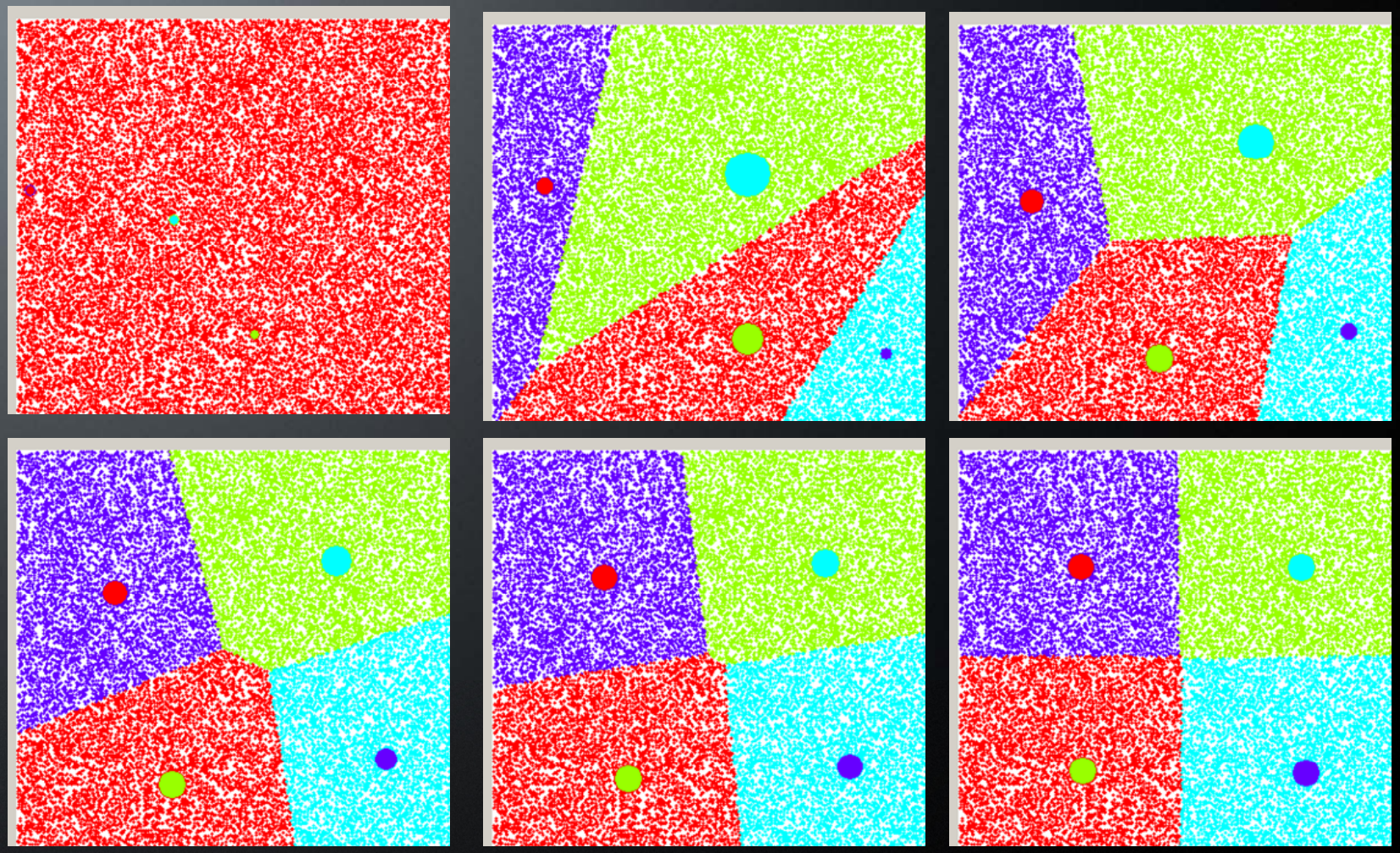
Key is the cluster, data is the object

Reduce: for all objects in the same cluster,

find the new cluster centers

Repeat till no object changes clusters

K means demo



K means performance experimental data

CPU 4 - core i7 processor

GPU (2 NV 260 mid range GPU's)

GPU programmed in CUDA

CPU programming in TBBU

(joint work with Balaji Dhanasekaran, U of Virginia)

AMD has announced a new higher performance GPU card (58xx series), which supports the industry standard language OpenCL.

Performance numbers using OpenCL on AMD hardware

Will be posted at http://developer.amd.com/Pages/default.aspx

Performance
4 million objects, 2 attributes, 100 clusters
50 iterations

Map	Reduce	Time (sec)	4 CPU	1CPU
2GPU	2GPU	.81	40.9	194
2GPU	4CPU	1.29		
1GPU	1GPU	1.54	21.5	102
1GPU	4CPU	1.71		
4CPU	4CPU	33.11	1	4.75
1CPU	1CPU	157.34		1

Measured performance for other sizes shows similar results, better for larger attributes, objects

AMD
The future is fusion

K-means

Do most of the work on the GPU, but do some of the reductions on the CPU if you have enough idle cores.

Preliminary numbers suggest GPU can solve this very well.

Can we develop software that automatically adjusts for this, or do programmers have to write variant algorithms?

Disclaimer and Attribution

Questions?

AMD
The future is fusion