[simula . research laboratory]

UNIVERSITY OF OSLO

Distributed Real-Time Processing of Multimedia Data with the P2G Framework

Paul B. Beskow, Håvard Espeland, Håkon K. Stensland, Preben N. Olsen, Ståle Kristoffersen
Espen A. Kristiansen, Carsten Griwodz, Pål Halvorsen
Simula Research Laboratory, Norway and IFI, University of Oslo, Norway

P2G is a framework designed to integrate concepts from modern batch processing frameworks into the world of real-time multimedia processing, where we seek to scale transparently with the available resources. P2G consists of a compiler and run-time that analyzes dependencies dynamically and merges or splits kernel instances based on resouce availability and performance monitoring.

KERNEL LANGUAGE

fetch <field F> (age A) [index X] ... [index Z]

Extract a slice of data from a field

store <field F> (age A) [index X] ... [index Z]

Store a slice of data to a field

P2G Kernel language Example

Field definitions:

```
0  int32[] m_data age;
1  int32[] p_data age;
```

Kernel definitions:

```
0  init:
1    local int32[] values;
2
3    %{
4      int i = 0;
5      for( ;i < 5; ++i )
6      {
7        put( values, i+10, i );
8      }
9    %}
10
11   store m_data(0) = values;
12
13
```

```
0  mul2:
1    age a;
2    index x;
3    local int32 value;
4
5    fetch value = m_data(a)[x]
6
7    %{
8      value *= 2;
9    %}
10
11   store p_data(a)[x] = value;
12
13
```

```
0  plus5:
1    age a;
2    index x;
3    local int32 value;
4
5    fetch value = p_data(a)[x]
6
7    %{
8      value += 5;
9    %}
10
11   store m_data(a+1)[x] = value;
12
13
```

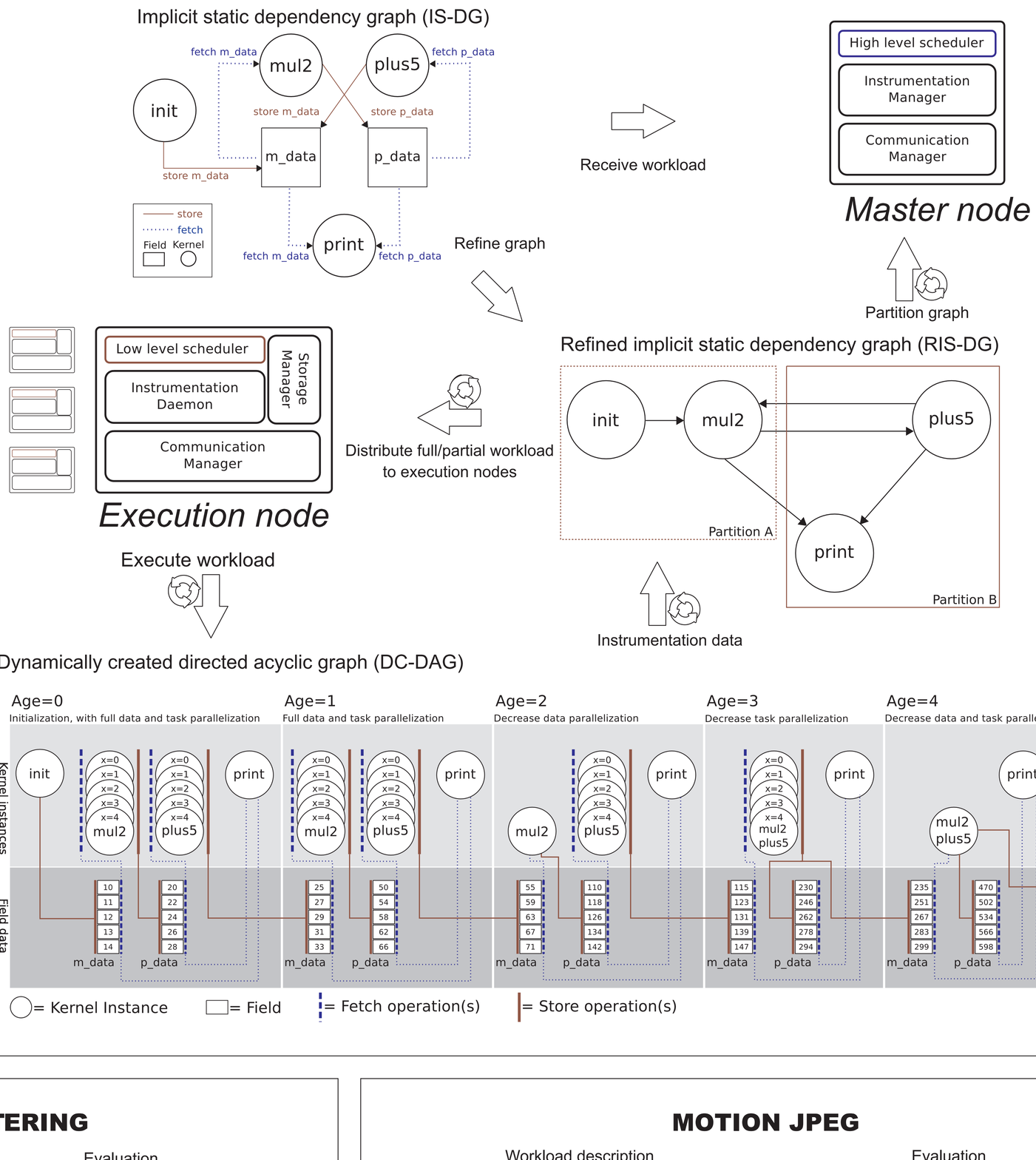
```
0  print:
1    age a;
2    local int32[] p, m;
3
4    fetch p = p_data(a)
5    fetch m = m_data(a)
6
7    %{
8      for(int i=0; i < extent(p, 0))
9      {
10       cout << "p: " << get(p, i);
11       cout << "m: " << get(m, i);
12     }
13   %}
```

C++ Equivalent

```
void print( int * data, int num )
{
  for( int i = 0; i < num; ++i)
    std::cout << data[i] << " ";
  std::cout << std::endl;
}

int main()
{
  int data[] = { 10, 11, 12, 13, 14 };
  int num = (sizeof data/sizeof *data);
  print( data, num );
  while(true)
    {
      for( int i = 0; i < num; ++i)
        {
          data[i] *= 2;
        }
      print( data, num );
      for( int i = 0; i < num; ++i)
        {
          data[i] += 5;
        }
      print( data, num );
    }
  return 0;
}
```

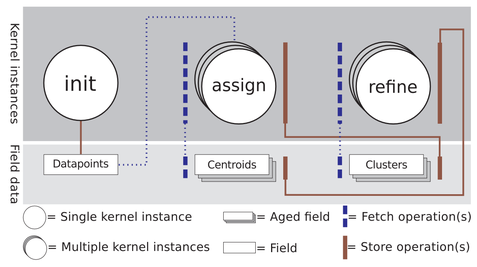
P2G ARCHITECTURE



K-MEANS CLUSTERING

Workload description

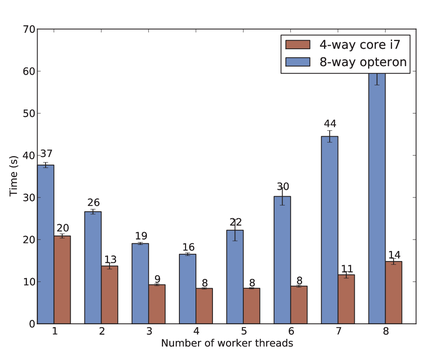
K-means clustering is an iterative algorithm for cluster analysis, which aims to partition *n* datapoints into *k* clusters in which each datapoint belongs to the cluster with the nearest mean.



The **init** kernel generates *n* datapoints, selecting *k* initial centroids. This results in *n* **assign** kernel instances per age, each fetching a single datapoint and the set of centroids for that age. It then calculates which of these centroids the point is closest to and stores the point in the cluster of that centroid (for the given age). *k* **refine** kernels are then dispatched (per age), each calculating the new mean of that cluster of datapoints, and then stores the result in the Centroids field. This process is then repeated until the algorithm converges.

Evaluation

The *k*-means workload was run on a generated dataset of *n*=2000 with *k*=100. The algorithm was not run until convergence, but stopped after 10 iterations, as the point of convergence is not deterministic.



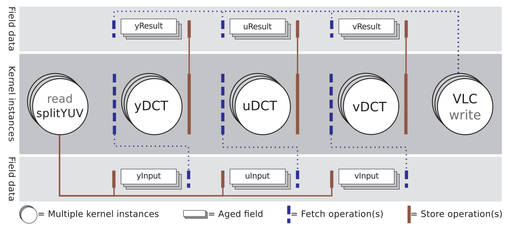
Microbenchmark

Kernel	Instances	Dispatch Time	Kernel Time
init	1	58.00 µs	9829.00 µs
assign	2024251	4.07 µs	6.95 µs
refine	1000	3.21 µs	92.91 µs
print	11	1.09 µs	379.36 µs

MOTION JPEG

Workload description

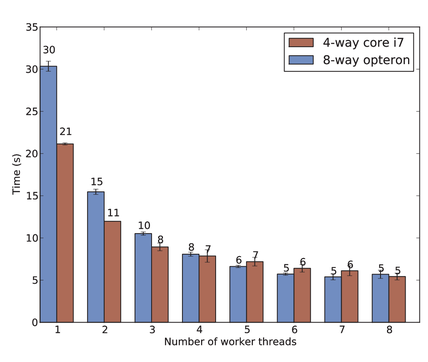
Motion JPEG (MJPEG) is a video coding format using a sequence of separately compressed JPEG images.



One **read/splitYUV** kernel instance is dispatched per frame read. It splits the image into its Y, U and V components and stores this to its corresponding fields. Then one kernel per macro block is dispatced, with CIF resolution (352*288) this equals 1584 **yDCT** kernels, 396 **uDCT** and **vDCT** kernels. These kernels then calculate the discreet cosine transform and do quantization, storing their results to their corresponding fields. Then one **VLC/write** kernel per frame is dispatched to perform variable length coding and writing the buffer to disk.

Evaluation

The MJPEG workload was run on the standard test sequence Foreman encoded in CIF resolution. We limited the workload to process 50 frames of video. A single-thread native implementation completed in ~30 sec on the Opteron and ~19 sec on the Core i7.



Microbenchmark

Kernel	Instances	Dispatch Time	Kernel Time
init	1	69.00 µs	18.00 µs
read/splitYUV	51	35.50 µs	1641.57 µs
yDCT	80784	3.07 µs	170.30 µs
uDCT	20196	3.14 µs	170.24 µs
vDCT	20196	3.15 µs	170.58 µs
VLC/write	51	3.09 µs	2160.71 µs