Pricing Python Parallelism

A Dynamic Language Cost Model for Heterogeneous Platforms

Dejice Jacob, Phil Trinder, Jeremy Singer

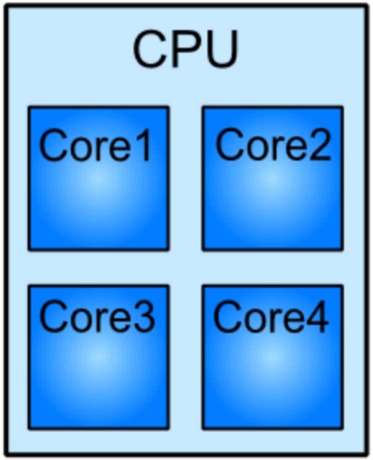
University of Glasgow

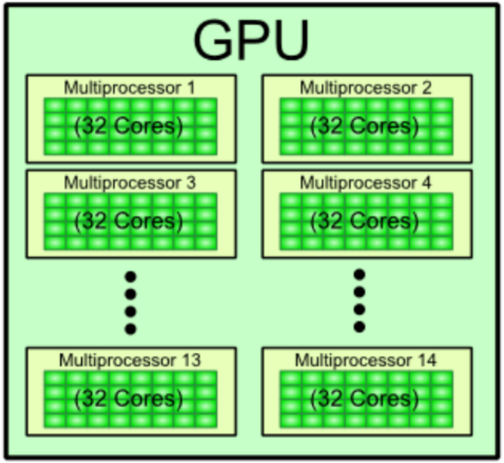
25/11/2020

University of Glasgow

School of
Computing Science

Heterogeneous architectures





ALPyNA

ALPyNA Novelties

- Staged parallelisation - Hybrid Static/Dynamic approach
- Preserving static analysis to aid runtime discovery of parallelism
- Runtime introspection of types and dependences
- Automatic loop parallelisation in a dynamic language

Novelties of ALPyNA Cost Model (ACM)

- Analytical cost model.
- Parametric – should account for differing hardware characteristics.
- Dynamic – sequential/parallel code structure can change.
- Light weight – JIT environment does not tolerate prediction latency.

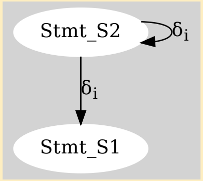
Runtime dependence analysis

```
import numpy as np

def ln_func(arg_a,k,limits) :
 im, jm = limits
 for i in range(0,im,1):
   for j in range(0,jm,1):
     # Statement - S1
     arg_a[i+k,j] = arg_a[i,j] + 4
     # Statement - S2
     arg_a[i+16,j] = arg_a[i,j]
```

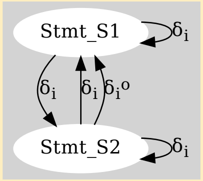
Runtime dependence analysis

```
import numpy as np

def ln_func(arg_a,k,limits) :
 im, jm = limits
 for i in range(0,im,1):
   for j in range(0,jm,1):
     # Statement - S1
     arg_a[i+k,j] = arg_a[i,j] + 4
     # Statement - S2
     arg_a[i+16,j] = arg_a[i,j]
```

$(im,jm) \leftarrow (32,1024)$ and $(k) \leftarrow 64$



$(im,jm) \leftarrow (32,1024)$ and $(k) \leftarrow (8)$



ACM Interpreter and CPU modelling

- Absolute cost model: predicts runtime
- Relative cost model: compares runtimes

Interpreter loop nest

$$C_{\text{int}}(s) = I_{\text{int}}(s) \prod_{f \in \mathcal{D}(s)} \mathcal{L}(f)$$

$$T_{\text{int}}(f) = \sum_{s \in \mathcal{E}(f)} C_{\text{int}}(s)$$

CPU loop nest

$$C_{\text{cpu}}(s) = I_{\text{cpu}}(s) \prod_{f \in \mathcal{D}(s)} \mathcal{L}(f)$$

$$T_{\text{cpu}}(f) = \sum_{s \in \mathcal{E}(f)} C_{\text{cpu}}(s)$$

GPU - SIMT Architecture

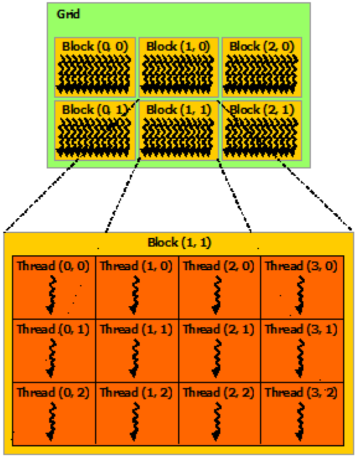


Figure: source:NVIDIA

Modelling GPU execution

Loop identification and splitting

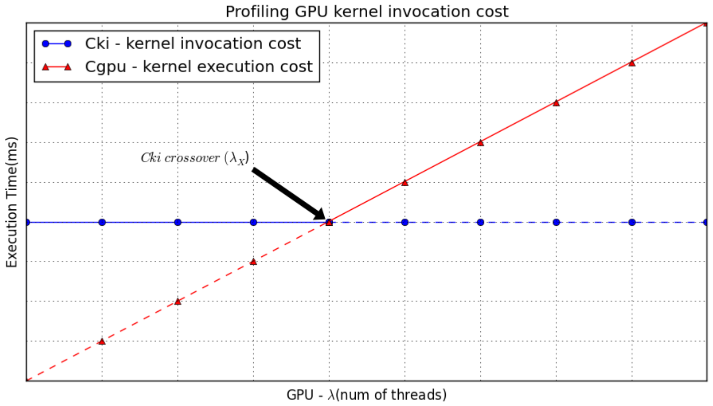
- loops executed sequentially on CPU: $\mathcal{D}_{seq}(s)$
- loops executed on GPU: $\mathcal{D}_{par}(s)$
    - loops executed with parallel threads: $\mathcal{D}_{gpu}(s)$
    - loops executed sequentially within kernels: $\mathcal{D}_{\overline{gpu}}(s)$

Cost of parallel execution on GPU

$$\lambda_{exec}(s) = \left\lceil \frac{g}{u} \right\rceil \times \frac{1}{g.v.w} \times \prod_{f \in \mathcal{D}_{gpu}(s)} \mathcal{L}(f) \times \prod_{f \in \mathcal{D}_{\overline{gpu}}(s)} \mathcal{L}(f)$$

$$g = \prod_{f \in \mathcal{D}_{gpu}(s)} \mathcal{G}(\mathcal{L}(f))$$

Accounting for GPU invocation



Smaller GPU workloads are executed faster on the GPU than the interpreter can dispatch a new kernel; the GPU is starved for work.

Calibration to calculate relative cost

Ideal compiled CPU execution relative to VM

$$\mu = \frac{l_{\text{int}}(s)}{l_{cpu}(s)} \tag{1}$$

Relative cost of GPU execution incorporates terms to account for CPU and (shared) GPU caches

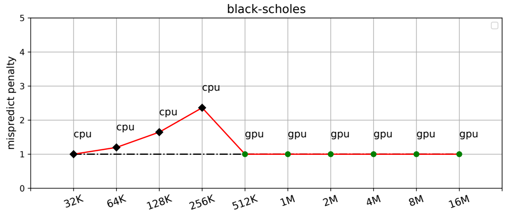
$$\frac{l_{cpu}(s)}{l_{\text{gpu}}(s)} = \psi \approx \frac{f_{gpu} \times (LC_{\text{gpu}}/\sigma)}{f_{cpu} \times LC_{cpu}} \tag{2}$$

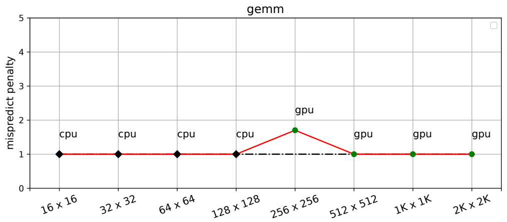
σ – cache sharing factor for GPU. CPU is single threaded.

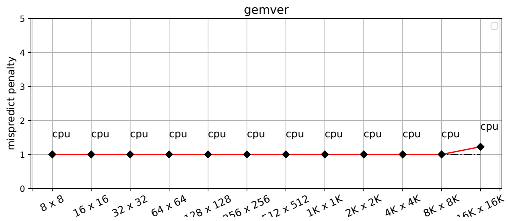
Derived GPU per core performance

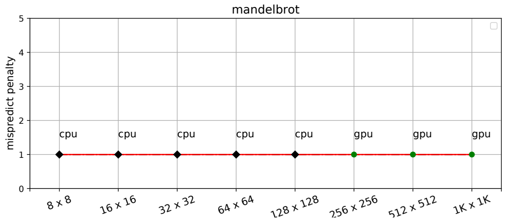
$$\frac{l_{\text{int}}(s)}{l_{\text{gpu}}(s)} \approx \mu \times \psi \tag{3}$$

Experimental Results – prediction performance

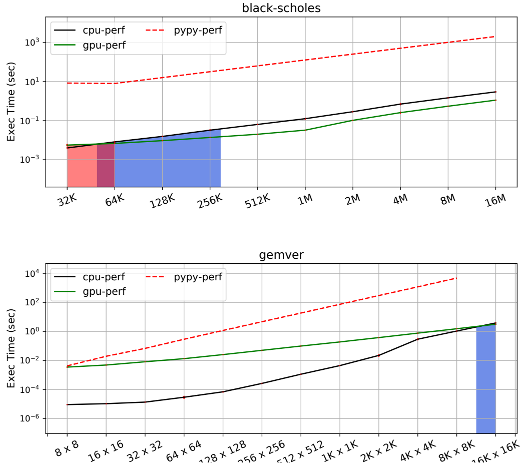


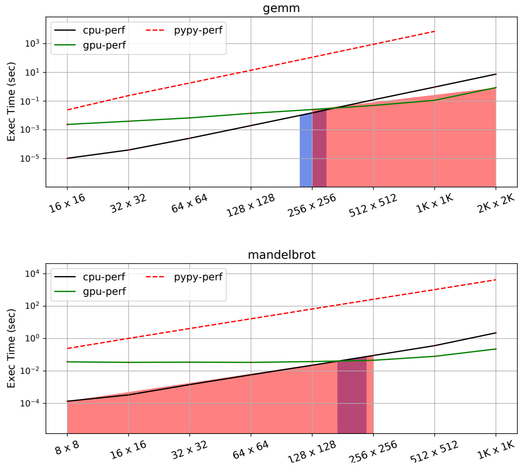






Experimental Results – misprediction range





Conclusion

1. A lightweight analytical cost model to select the faster compute device for a loop nest in a heterogeneous architecture.
2. Adapts to runtime dependence analysis and code generation.
3. Minimal install time profiling required
4. Overall 13.6% mean slowdown due to mispredictions.
5. Overall 14.3% mean misprediction across iteration domain sizes

Conclusion

Publications

- DLS–20 – doi:10.1145/3426422.3426979
- DLS–19 – doi:10.1145/3359619.3359743
- ARRAY–19: doi:10.1145/3315454.3329956
- Source: `https://bitbucket.org/djichthys/alpyna/src/master`

UKRI Engineering and Physical Sciences Research Council