PARMA: Parallelization-Aware Run-time Management for Energy-Efficient Many-Core Systems

Newcastle PRIME team, IEEE TC, 69(10), Oct 2020.

Parallelization and runtime

- Multiple cores in the h/w
- s/w of different degrees of parallelizability
- How to obtain optimal runtime decisions with regard to task to core mapping?

Intuition and hypothesis

- If an application is not parallelizable, giving it multiple cores would be wasteful
- If an application is parallelizable, giving it a single core does not exploit the h/w fully
- It is therefore reasonable to expect that runtime decisions based on the parallelizability of apps may lead to energy/performance optimimality

What does parallelizable mean?

- Amdahl's Law
  - "In computer architecture, Amdahl's law is a formula which gives the theoretical speedup in latency of the execution of a task at fixed workload that can be expected of a system whose resources are improved."

$$S(n,1) = \frac{I \cdot t(1)}{I \cdot t(n)} = \frac{1}{(1-p) + \frac{p}{n}}$$

What does parallelizable mean?

- Amdahl's Law
  - Larger p – more parallelizable
  - Smaller p – less parallelizable
  - The simplicity of Amdahl's Law makes it suitable for runtime use

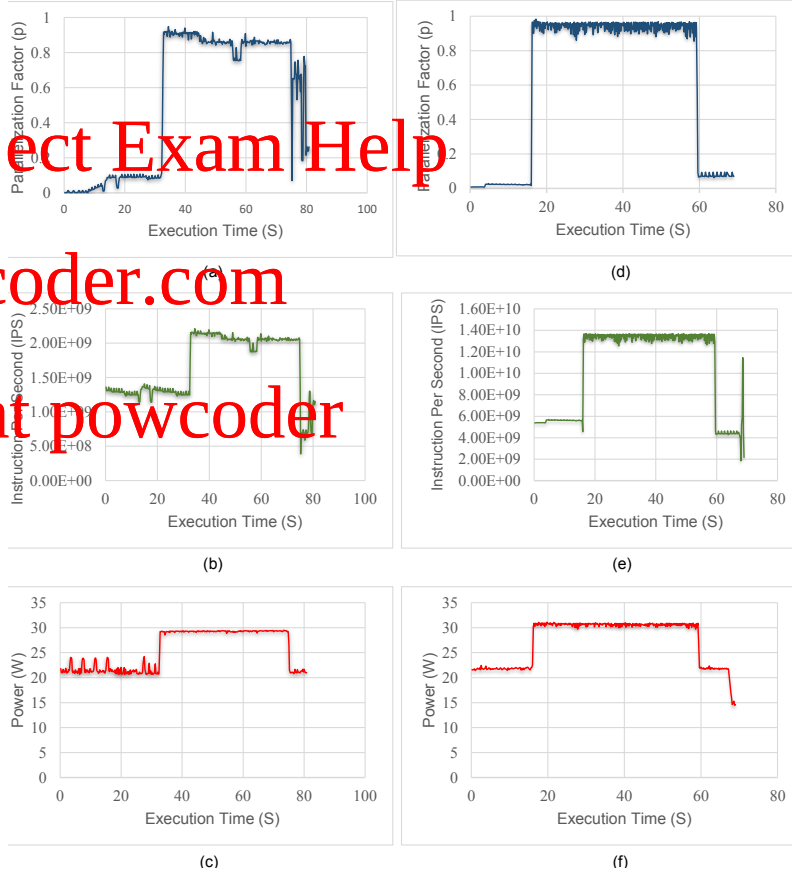
$$S(n,1) = \frac{I \cdot t(1)}{I \cdot t(n)} = \frac{1}{(1-p) + \frac{p}{n}}$$

But how can this be used for runtime?

- The runtime control needs the following inputs
  - App parallelizability (the *p* factor)
  - Availability of cores
- It makes the following decision
  - Map each app to the optimal number of cores

So each app has a *p* value?

- Not so simple
  - Apps may have phases during their execution
  - One *p* per app is not optimal



Runtime *p* factor sensing

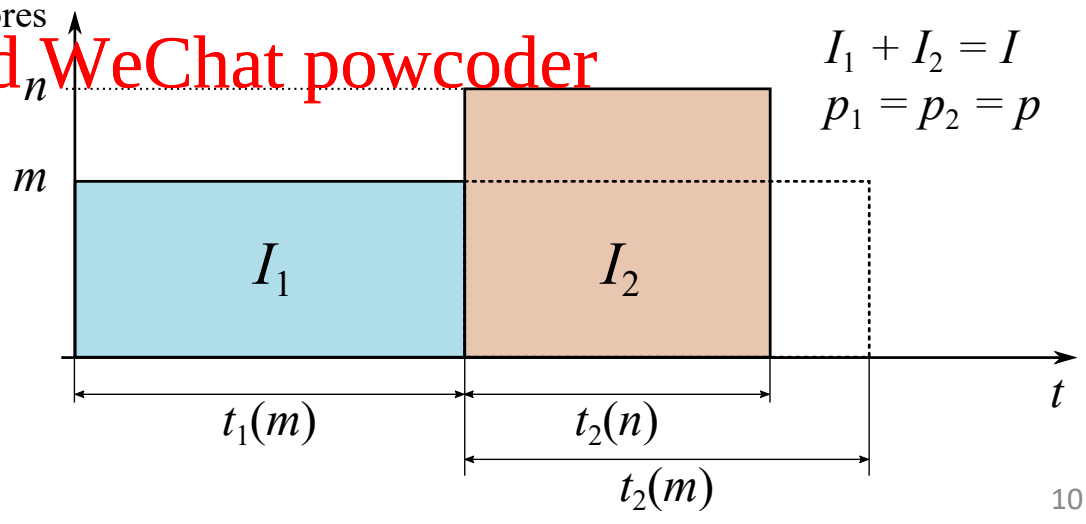
- If we could determine the *instantaneous p* value for each app
  - We'd be able to schedule it optimally on a per control cycle basis
  - So the idea is to find the *p* value of an app for each control cycle and make runtime decisions based on that

But how?

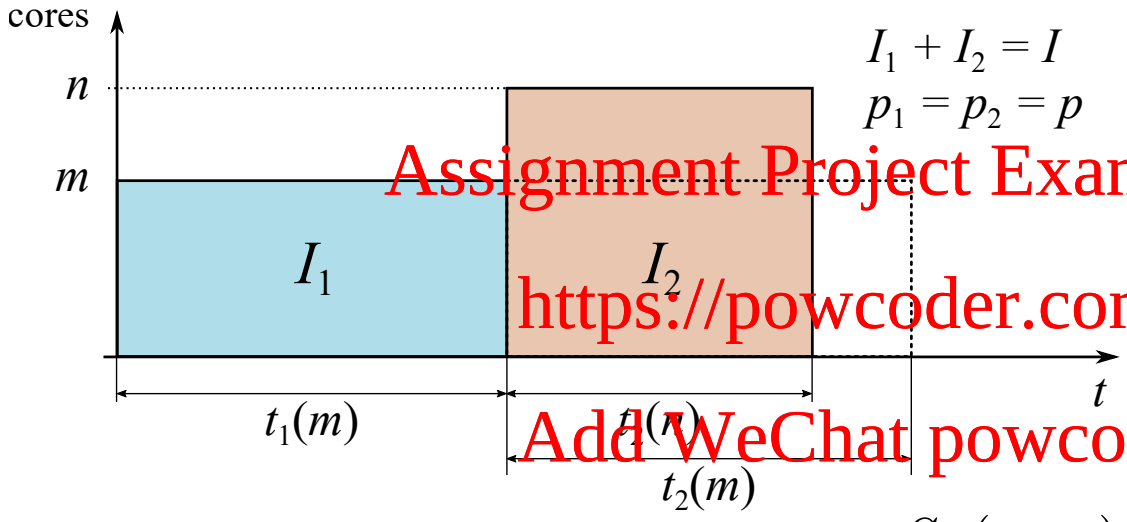
- Usually you can find the *p* value of an app in offline static characterization by comparing relative speedup between runs on different numbers of cores
  - Using Amdahl's Law backwards – knowing *S*, find *p*.
  - This is not practical at runtime on a per control cycle basis

Not the entire app

- An app can be run on two different core configurations for very small amounts of time and the relative speedup calculated from the observations



Calculating *p*

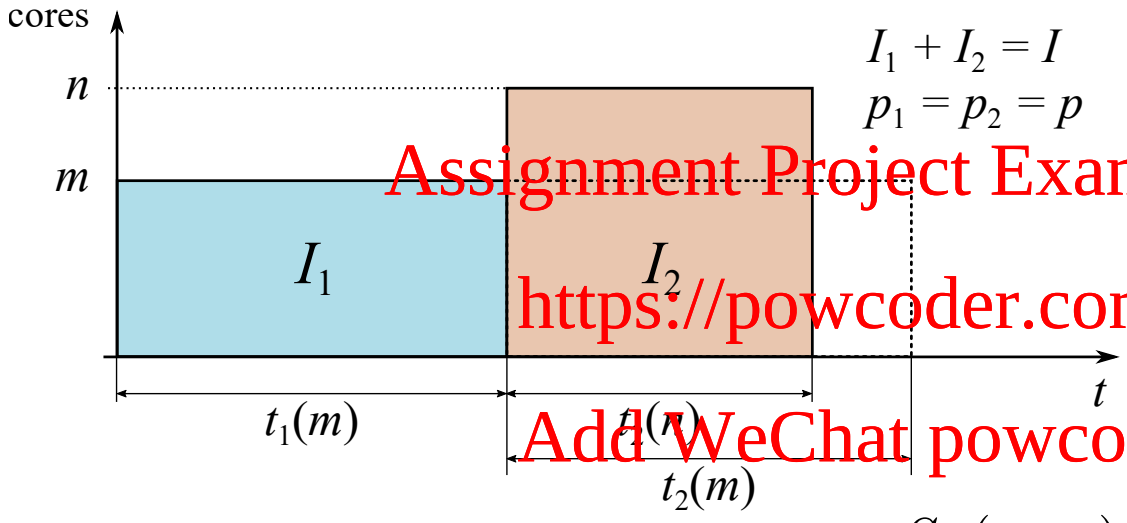


$$I_1 + I_2 = I$$
$$p_1 = p_2 = p$$

$$S_2(n,m) = \frac{I_2 \cdot t_1(m)}{I_1 \cdot t_2(n)} = \frac{IPS_2(n)}{IPS_1(m)}$$

$$p = \frac{S(n,m) - 1}{S(n,m) \cdot \frac{(n-1)}{n} - \frac{(m-1)}{m}}$$

Calculating *p*



$$I_1 + I_2 = I$$
$$p_1 = p_2 = p$$

IPS can be obtained through PMC's

$$S_2(n,m) = \frac{I_2 \cdot t_1(m)}{I_1 \cdot t_2(n)} = \frac{IPS_2(n)}{IPS_1(m)}$$

$$p = \frac{S(n,m) - 1}{S(n,m) \cdot \frac{(n-1)}{n} - \frac{(m-1)}{m}}$$

Optimization objective functions

- You can give performance (throughput) and power different degrees of importance
- Multiple factors in the optimization objective
  - Weighted sum – seems intuitive, but in this case, difficult to argue for (performance and power are not in the same dimension so weights tend not to make a lot of sense
  - Weighted product – this makes sense when the multiple factors are different types of physical quantities, but weights are not multiplied to the factors, they are powers

Optimization objective functions

- Examples:
  - PNP – power normalized performance in the form of IPS/Watt (how much performance you can get from spending a watt of power) – this has the dimension of 1/energy (1/J)
  - This is the inverse of energy per operation, so both describe the same optimization target, maximizing one minimizes the other
  - EDP – energy-delay product, energy per operation multiplied to latency per operation, this puts more emphasis on speed than those above, to minimize this you maximize $(IPS)^2/Watt$
  - You can arbitrarily have E^xDP with x being any real number (usually greater than 0), E^2DP is essentially IPS/Watt

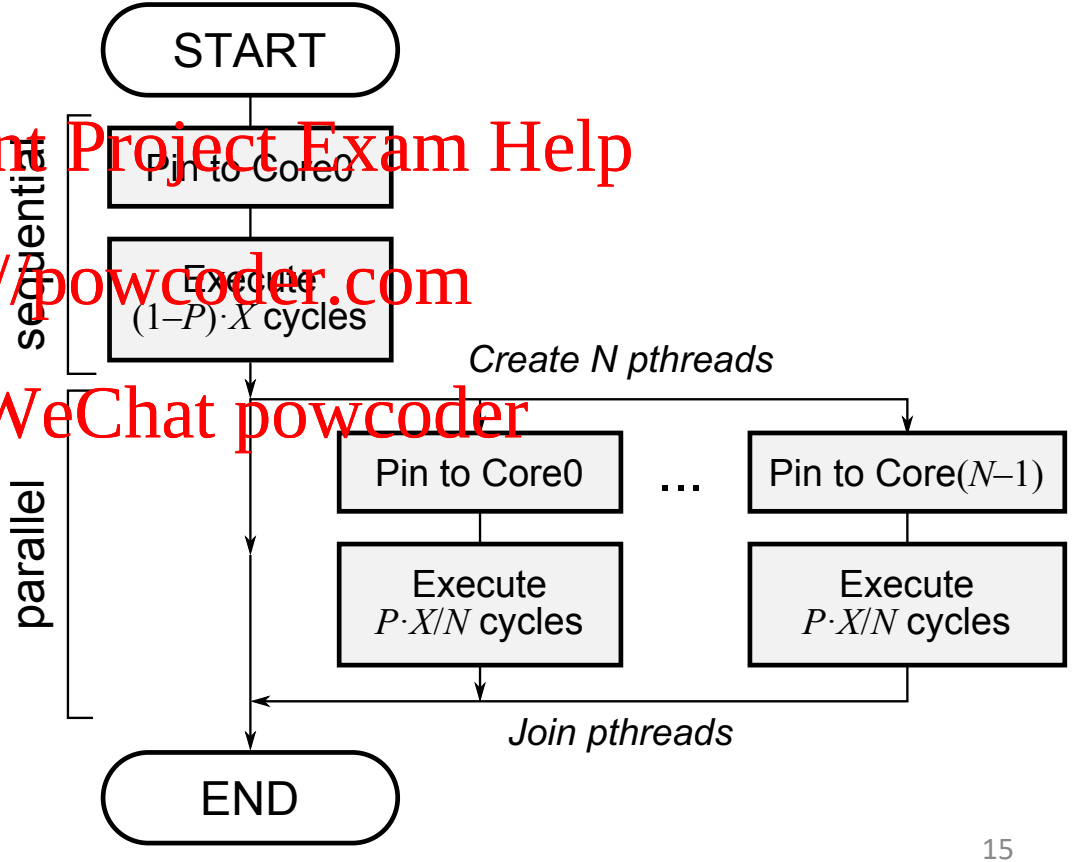
But how to relate *p* to objectives?

Run pthreads to characterize system parameters with regard to the *p* factor

pthreads allows a *p* ratio to be set by the user

Also allows the running of an arbitrary number of cores and specific cores

Build a set of lookup tables that relate *p* to optimal control decisions



Runtime

- Once you have a way of sensing *p* and a map of optimal decisions for each *p* value, you can have the following: