Tomas Plachetka

Independent Task Placement and its Tuning in the Context of Demand-Driven Parallel Ray Tracing

Grenoble, June 11, 2004

University of Bristol
United Kingdom

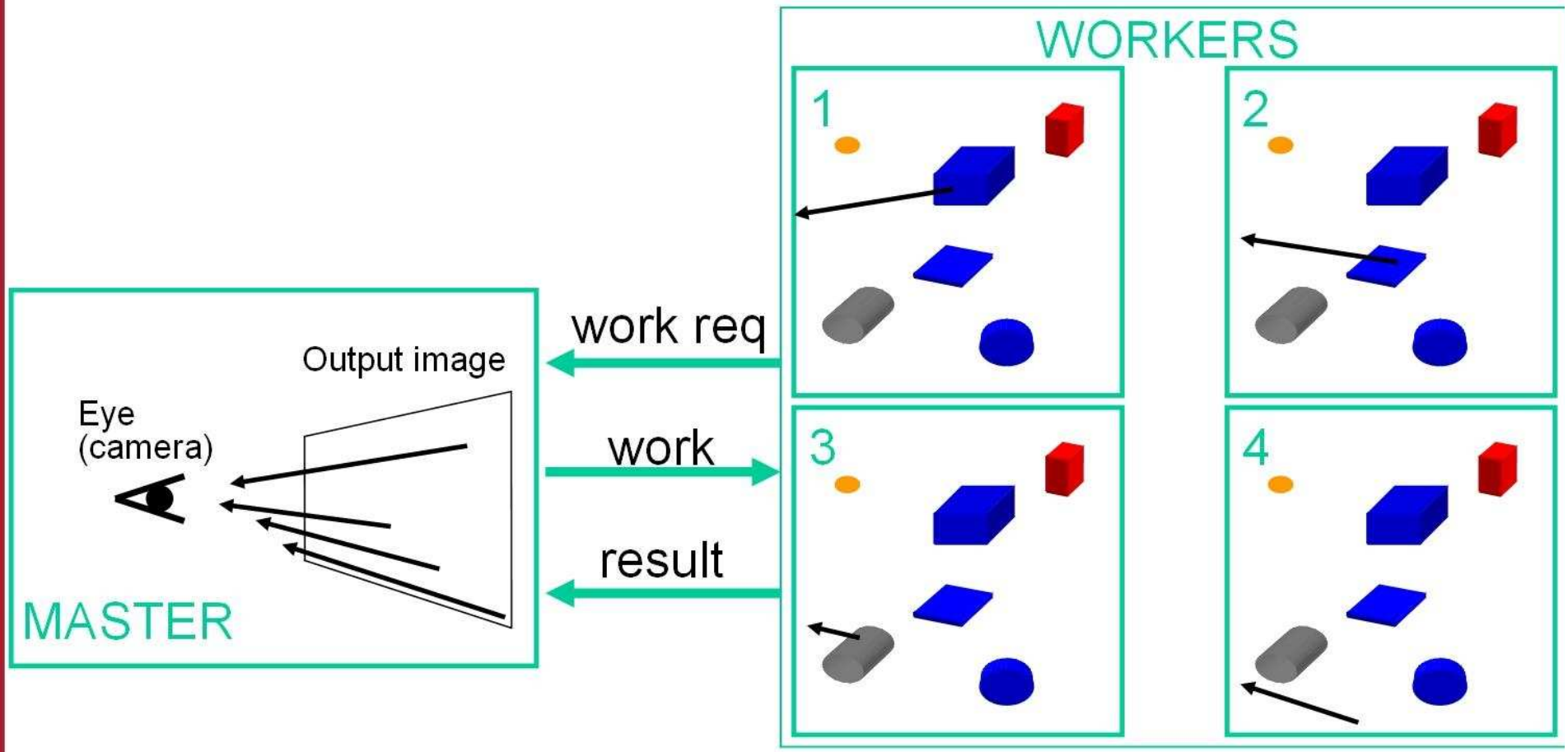
PC² Paderborn Center for Parallel Computing
University of Paderborn
Germany

Comenius University
Slovakia

Overview

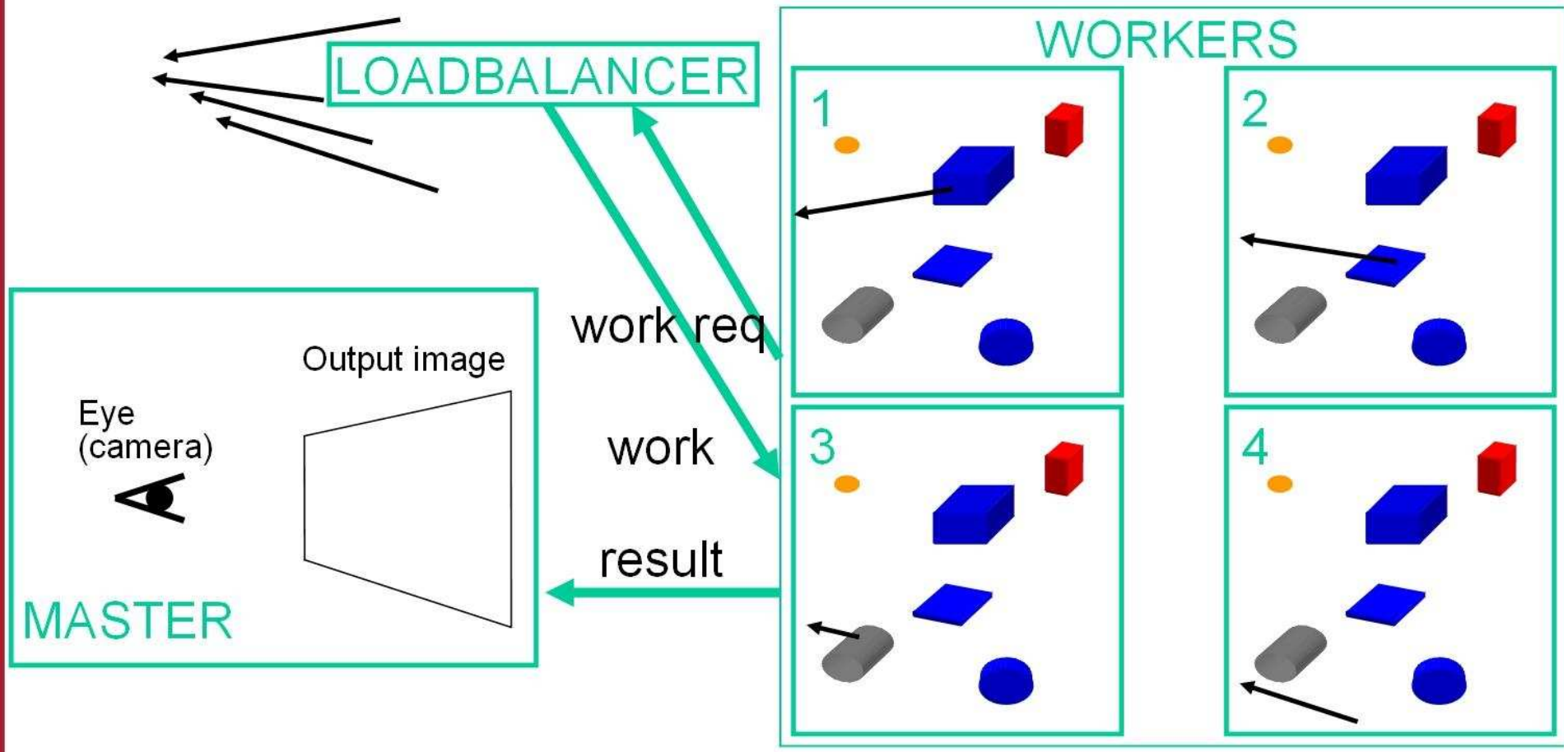
- Screen partitioning: demand-driven process farm
- Problem definition and examples of tradeoffs
- Analysis of chunking and factoring strategies
- Experiments with chunking and factoring, with manual tuning of parameters
- How efficient will your program be on your machine for a given number of processors for a given input?
- Is the choice of assignment strategy really important (in the context of contemporary parallel ray tracing)?

Parallel ray tracing: screen partitioning (**demand-driven process farm**)



Parallel ray tracing: screen partitioning (**demand-driven process farm**)

What does the LOADBALANCER process do?



Given:

N nr. of worker processes, all equally fast

W nr. of **tasks**, independent of each other (not even spatially coherent)

L latency; i.e. overhead of assigning 1 **job** to a worker (a **constant** time which does not depend on the number of tasks in 1 job or anything else)

Unknown:

Task time complexities.

Goal:

Minimise the makespan (the parallel time required for assigning & processing of all tasks). This problem is **online**, because the LOADBALANCER must make a decision as to how many tasks to pack into a job immediately after receiving a work request. Well, online… our W is constant!

Probabilistic model

μ average tasks' time complexity

δ std. dev. of tasks' complexities

Goal: minimise **expected makespan**

Deterministic model

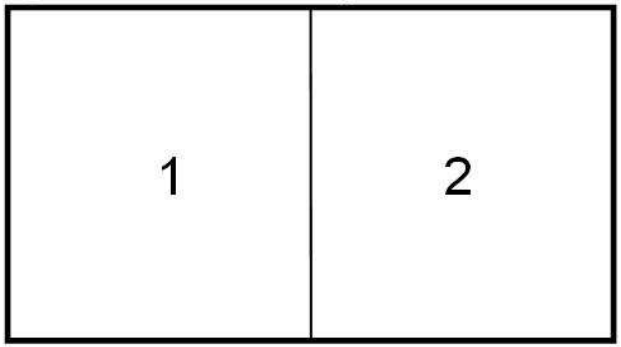
T_{max} maximal tasks' time complexity

T_{min} minimal tasks' time complexity

Goal: minimise **worst-case makespan** (for worst possible task arrangement)

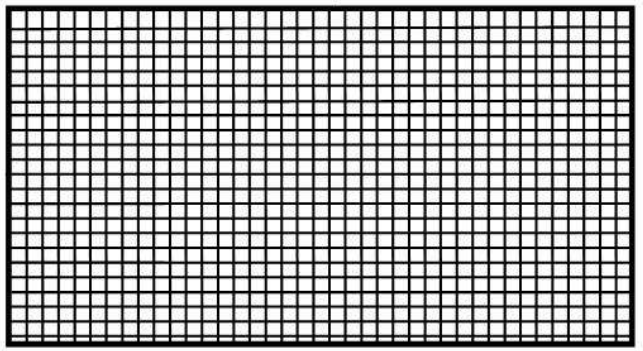
Chunking

Largest chunks (for 2 workers)



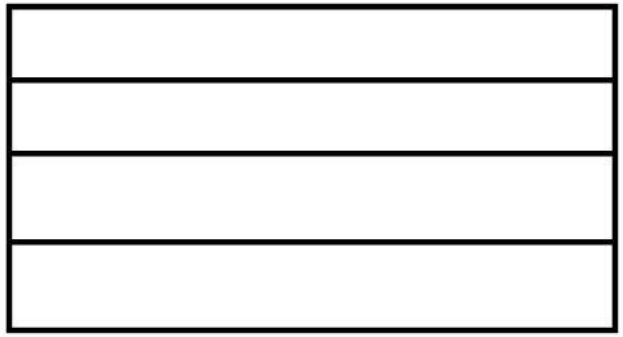
Problem: **imbalance**

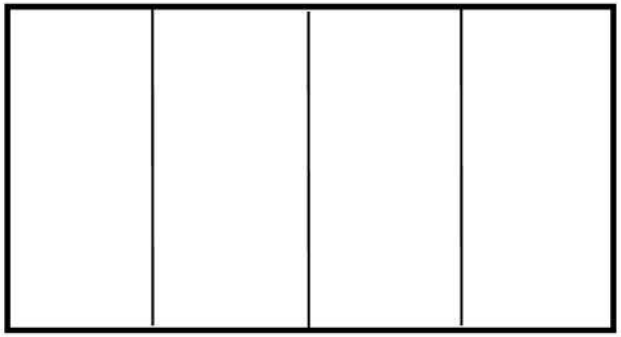
Smallest chunks (1 chunk=1 task)



Problem: **many messages**

The chunk "shape" is irrelevant





How large should the chunks be?

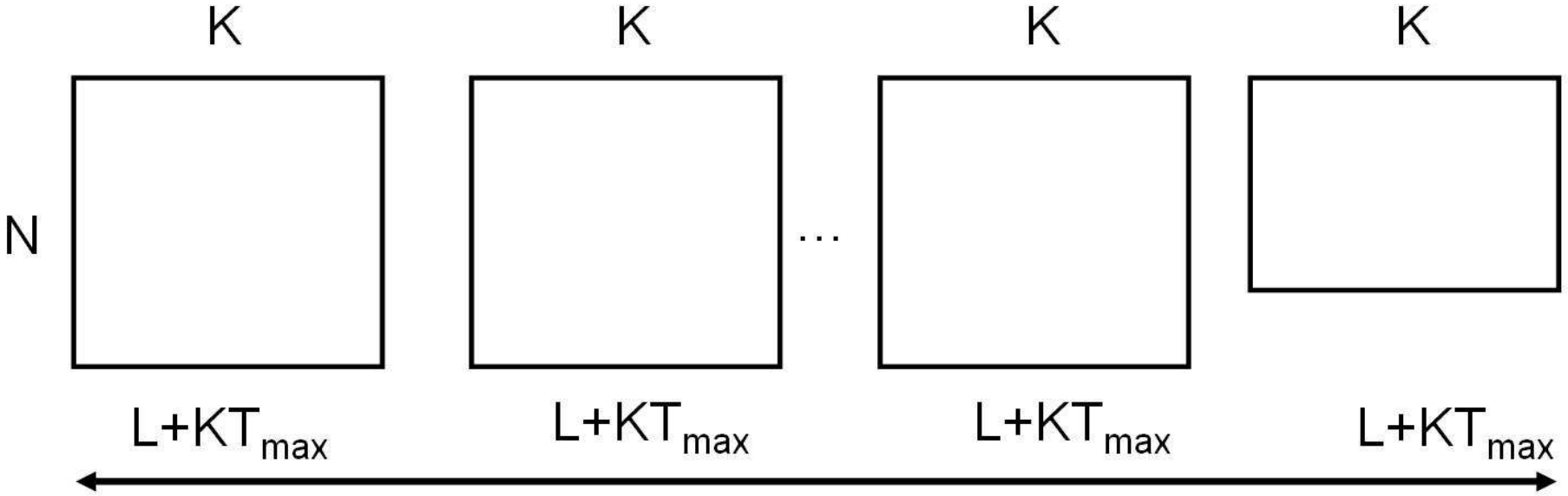
(… so that the worst-case makespan is minimal!)

Chunking, worst-case analysis

Parameterisation:
N nr. of workers
W nr. of tasks
L latency
T_{max} max. task complexity

Unknown: K_{opt} (chunk size)

The time diagram below depicts the structure of the worst case (worst makespan):
One of the workers always gets the tasks of time complexity T_{max}.



$$M = \left(1 + \frac{W}{NK}\right)(L + KT_{\max}) \qquad K_{opt} = \sqrt{\frac{WL}{NT_{\max}}} \rightarrow M_{opt} = \frac{WT_{\max}}{N} + L + 2\sqrt{\frac{WT_{\max}L}{N}}$$

$$M'(K) := 0$$

Factoring

$$3\ \frac{x}{4}W \quad \geq \quad \frac{3}{4}W \Rightarrow \mathbf{x \geq 1} \Rightarrow \mathbf{x = 1}$$

$$K = \max\left(1, \left\lfloor \frac{W_{rest}}{1 + T \cdot (N-1)} \right\rfloor\right)$$

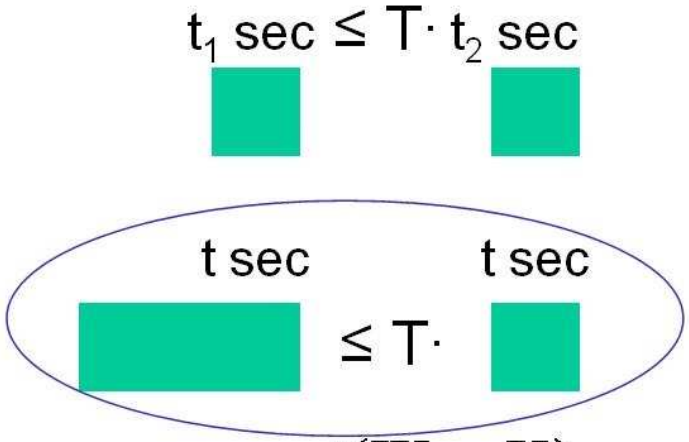
Parameterisation:

N nr. of workers

W nr. of tasks

T max. ratio of tasks' complexities ($T=T_{max}/T_{min}$)

Unknown: the job size K used for the next round



$$t(K) \leq \frac{t(W-K)}{N-1}$$

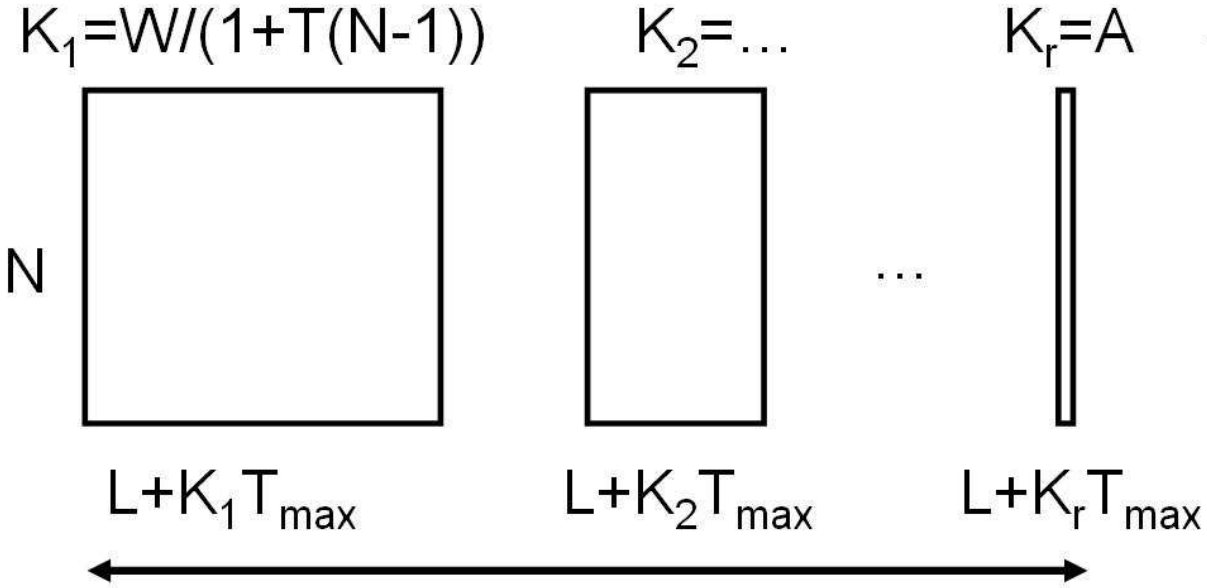
$$TK \leq \frac{W-K}{N-1}$$

Factoring, worst-case analysis

Parameterisation:

N nr. of workers
W nr. of tasks
L latency
T_{max} *max. task complexity*
T_{min} *min. task complexity*
$T=T_{max}/T_{min}$
A atomic job size (size of the smallest job)

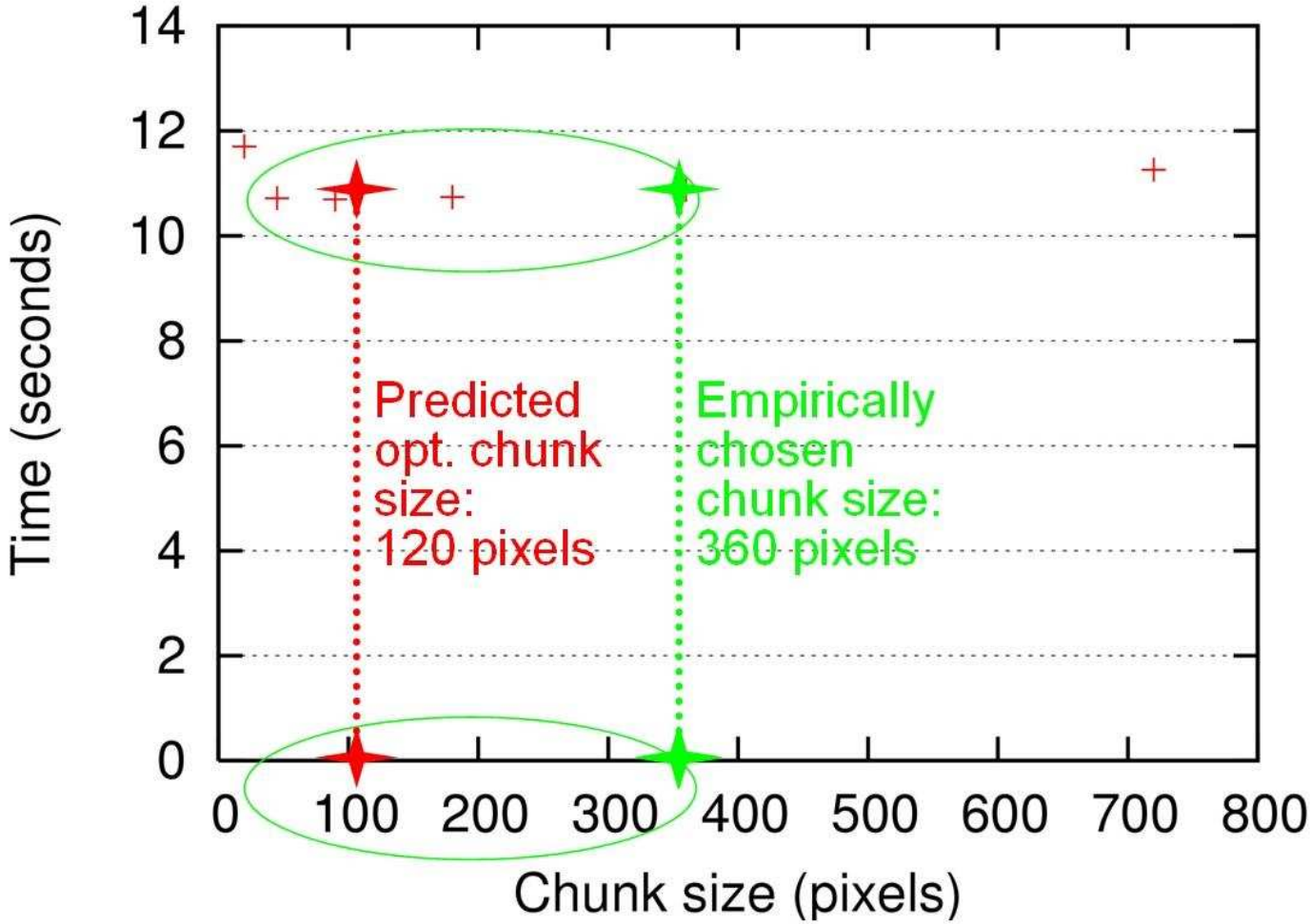
Structure of the worst case (time diagram below).
One of the workers always gets tasks of complexity T_{max}.



$$M_{opt} = \frac{(W-A)T_{\max}}{N} + Lr + L + AT_{\max} \qquad r = 1 + \left\lfloor \log_{1-\frac{N}{1+T(N-1)}} \left(\frac{AN}{W} \right) \right\rfloor$$

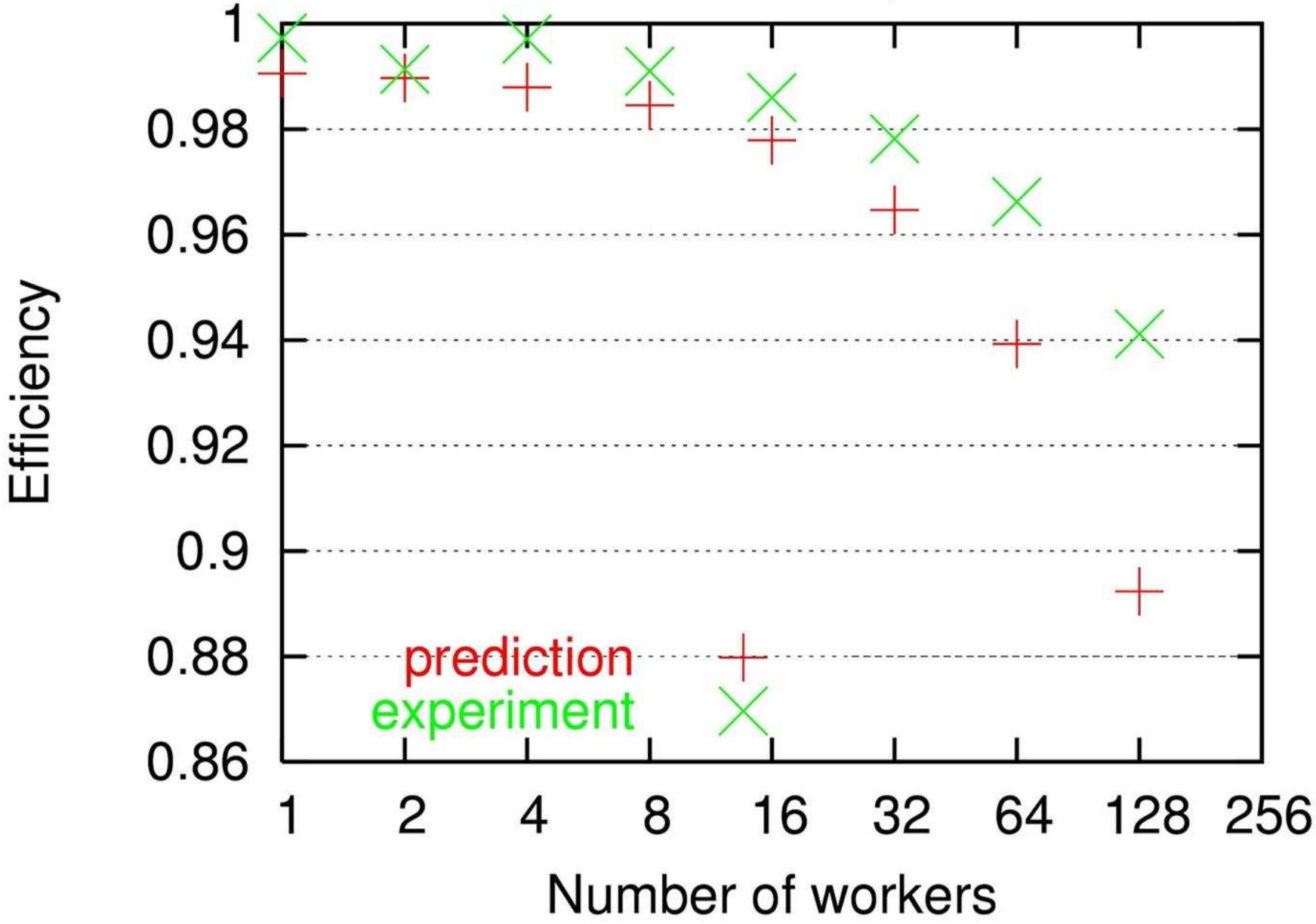
Screen partitioning, tuning of the chunk size

Manual tuning for 90 workers, POV||Ray

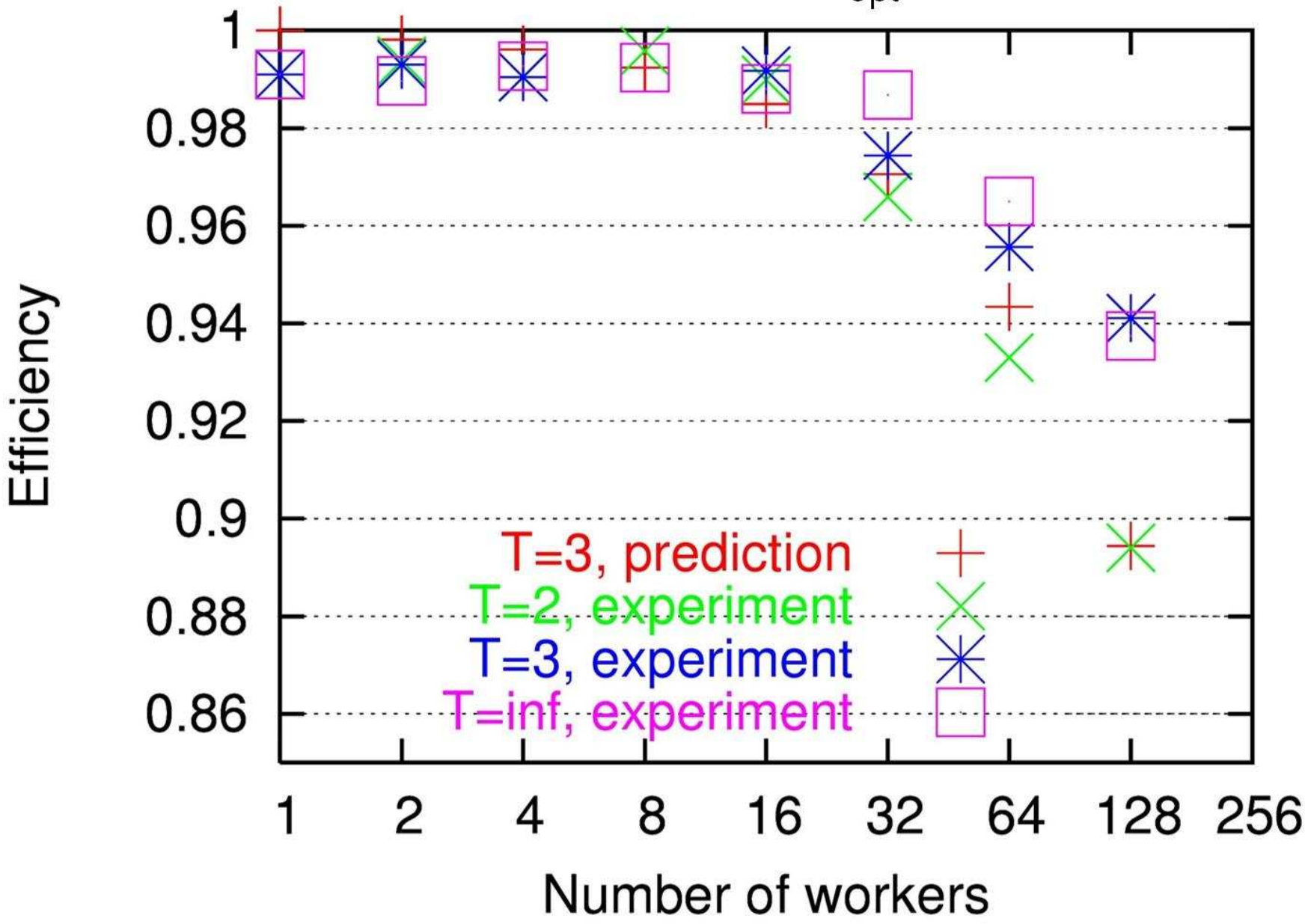


$$\hat{K}_{opt}^{CHUNKING} = \hat{A}_{opt}^{FACTORING} \approx 45 \ldots 360\ pixels$$

Screen partitioning, chunking with K_{opt}=360



Screen partitioning, factoring with A_{opt}=360



Tuning of factoring strategy (tuning of atomic job size *A*)

Communication time may exceed the communication time for one pixel. If it does then LB should always distribute more than one pixel in a job. The parameter *A* determines the size of the minimal job (in pixels).

LB: start stopwatch

LB: send work to a worker

LB: …

LB: receive another work request and T_{job} from the same worker

LB: stop stopwatch

T_{total}

$T_{comm} = T_{total} - T_{job}$

worker: receive work (P_{job} pixels)

worker: start stopwatch

worker: compute

worker: stop stopwatch

worker: send work request and report T_{job} to LB

T_{job}

Optimal setting of *A* (in LB process):

$A = \min(P_{job})$, such that $T_{comm} \leq T_{job}$

Tuning of factoring strategy (tuning of parameter T)

The optimal setting of the T is independent of the optimal setting of A.

The parameter T controls the trade-off between chunking (for $T \rightarrow \infty$) and static job assignment (for T=1). The optimum T_{opt} is somewhere between the two extremes.

An overestimation of T_{opt} ($T > T_{opt}$) leads to unnecessary communication.

An underestimation of T_{opt} ($T < T_{opt}$) leads to imbalance.

Solution 1 (optimistic):

A realistic (empirical) guess of T + work stealing phase.

No run-time tuning of T. T remains a constant during the farming phase.

Work stealing eliminates a possible imbalance.

The work stealing phase begins immediately after LB answers “no more work” to a work request (hence, work stealing overlaps with farming). In such a case the worker begins to actively look for work among other workers.

Tuning of factoring strategy (tuning of parameter T)

Solution 2 (conservative):

Run-time tuning of T (no work stealing phase).

This solution avoids an underestimation of T_{opt}.

In the first round, $T \rightarrow \infty$ is assumed (jobs of size A are distributed).

After a worker finishes, T is set to the maximum time ratio which has been observed until that moment (a pessimistic setting).

Despite the pessimistic setting, an underestimation can occur. However, LB detects this situation as some worker does not finish "in time" according to the current T (LB sets "deadlines" for workers according to its current setting of T).

If underestimation is detected, LB sends an ABORT signal to the late worker, making the worker finish a.s.a.p. with only a partial result. The pixels which have not yet been computed are returned back to LB. According to the number of the finished pixels in the partial result, LB adjusts the parameter T (pessimistically) and continues.

Conclusions

- Farming yields almost a linear speedup (efficiency 95% with 128 workers) for parallel ray tracing (POV||Ray) on a fairly complex "everyday" scene.
- Trivial chunking algorithm with optimal chunk size does not perform worse than a theoretically better factoring algorithm with optimal chunk size; for the particular machine, particular nr. of processors, particular input, particular quality settings, particular room temperature etc. used during these experiments.
- Efficiency of chunking/factoring can be predicted for a particular machine, particular nr. of processors, particular input, particular room temperature etc.
- In experiments with process farming, parameters W (nr of tasks), N (nr of workers), L (latency), T_{min} and T_{max} (min/max tasks' or jobs' time complexities) must be reported. Reporting only W, N and an empirical chunk size is insufficient for drawing conclusions from experiments with process farming (e.g. chunking).
- The parameters specific to chunking/factoring can (and must) be tuned automatically in run-time.