BRNO UNIVERSITY OF TECHNOLOGY | FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2023

logimic — We make IoT world

Optimizing KPI Processing of Smart Cities Using Multithreading in Node.js

Ondřej Šulc, xsulco01@vutbr.cz, Brno 2023

Key Performace Indicators

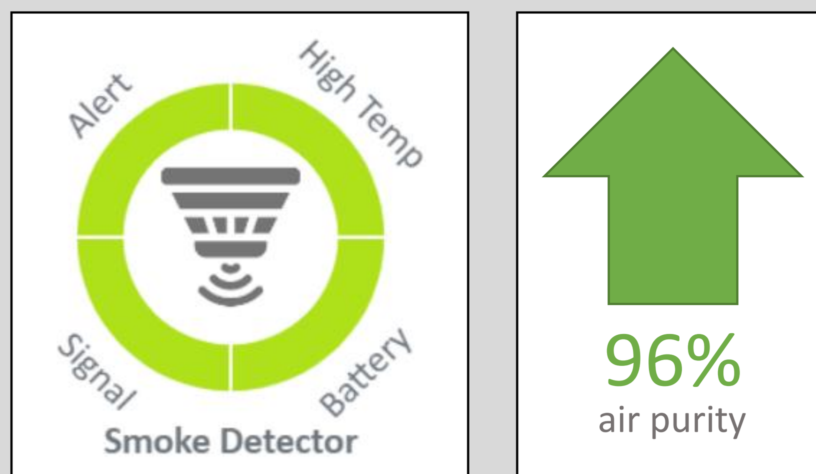

Figure 1: KPI output

Figure 1: The goal of KPI is to make the output as easy to read as possible – usually in the form of a control board containing the results of individual indicators.

Figure 2: This figure illustrates simple principle of converting data from IoT sensors into KPI. It can also be seen from this figure that the processing mostly consists of mutually independent steps.

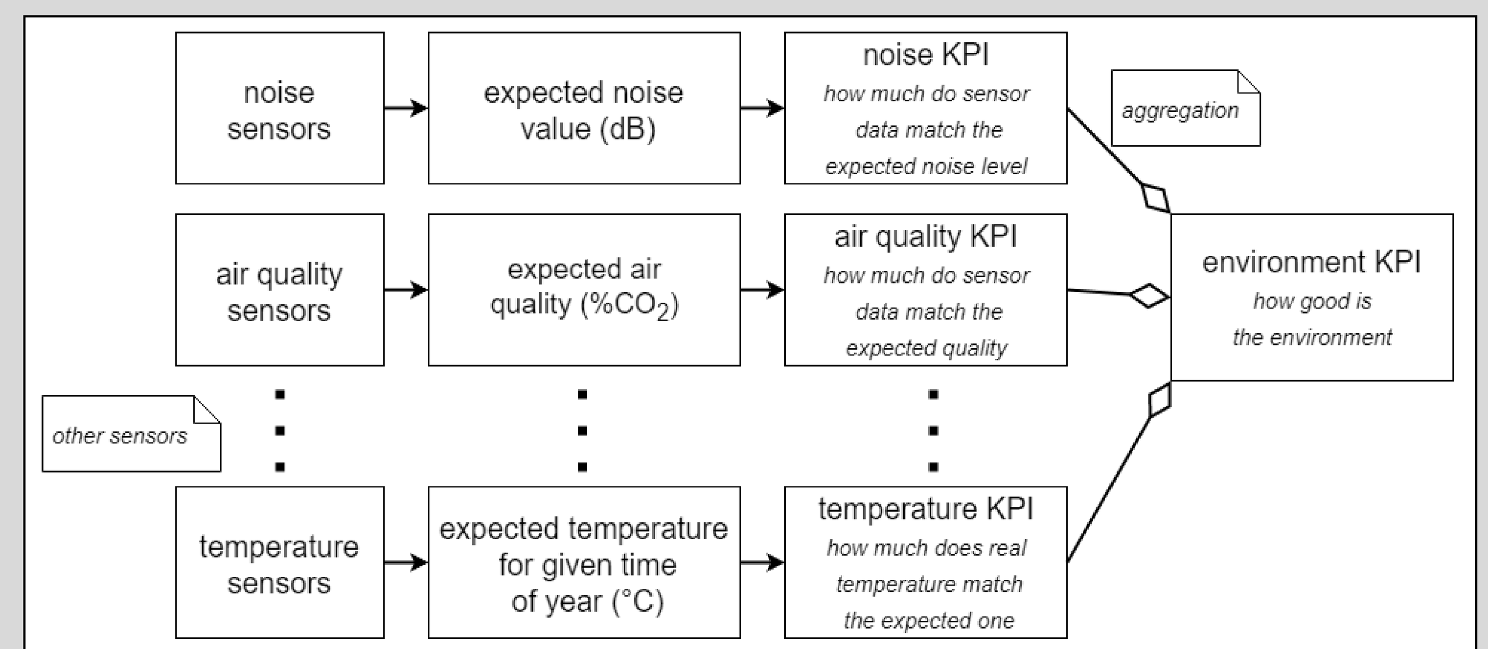

Figure 2: KPI from sensors

Multithreading

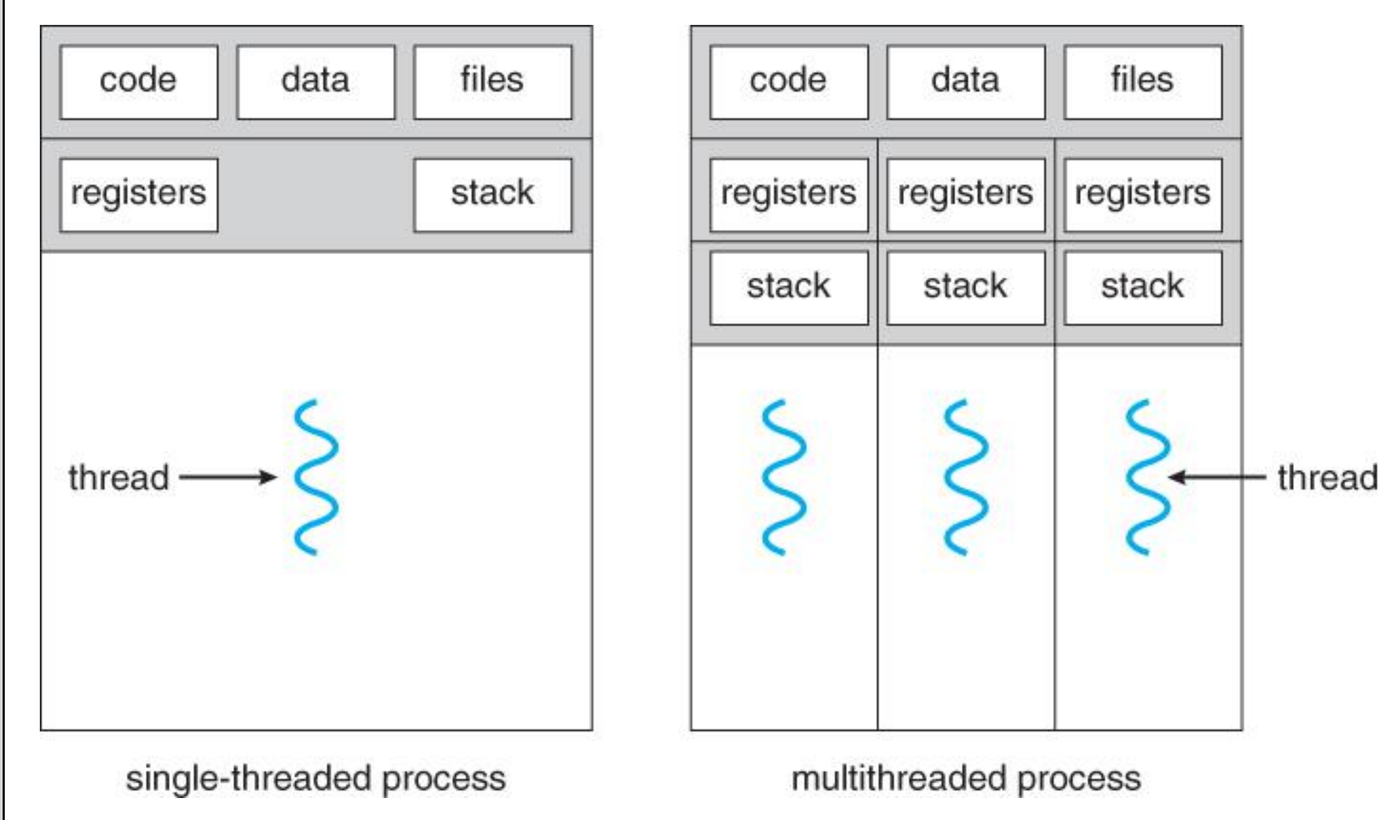

Figure 3: Multithreading [1]

Figure 4: The threads can be managed individually (that is, the main thread directly determines which thread will perform which work) or a shared work queue can be created, whereby the required actions will then be evenly distributed between the individual threads.

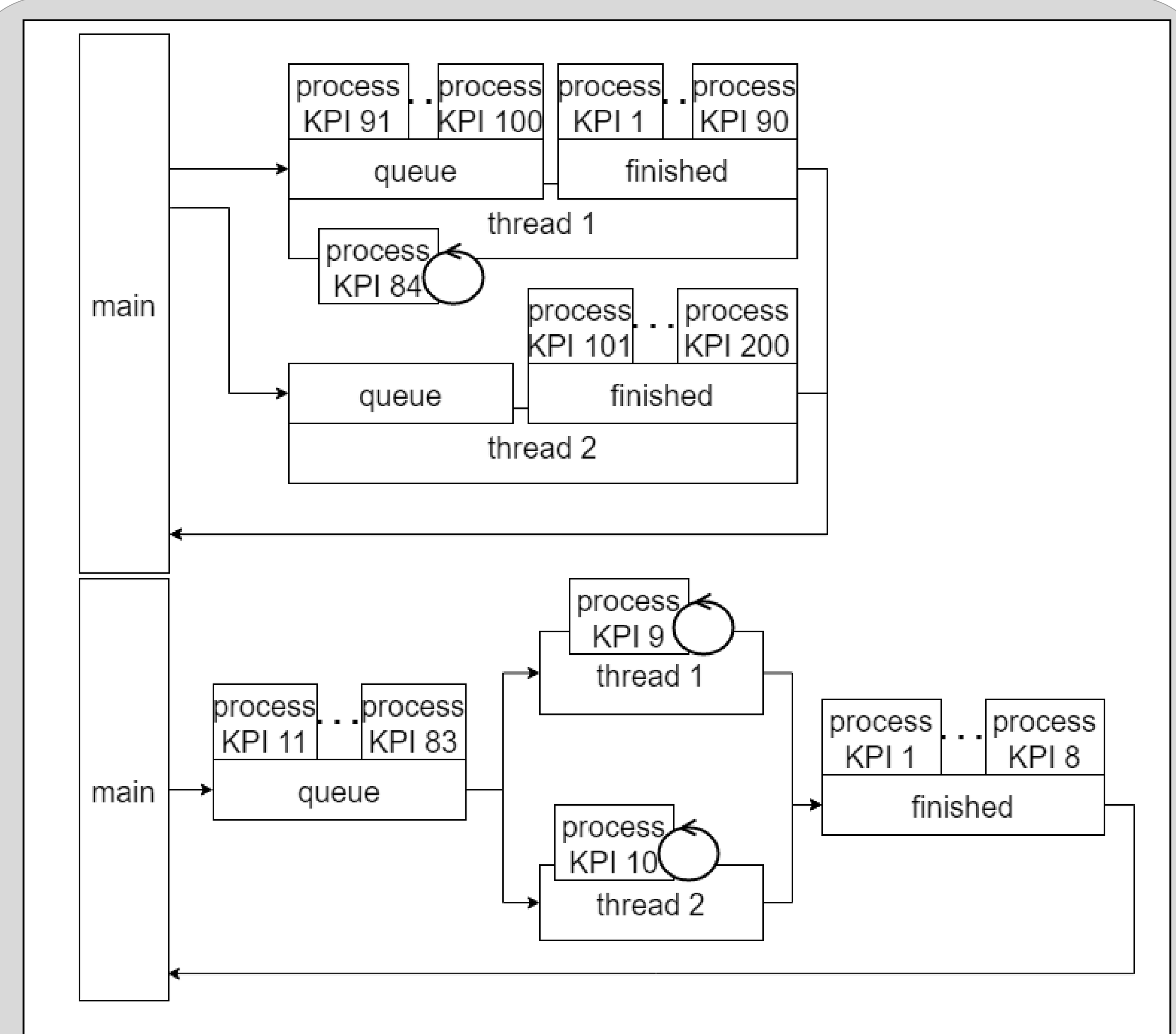

Figure 4: Threads management

Figure 3: Multithreading allows the application to use more processor cores. Threads share source code, data from before the threads were created, and access to files. On the other hand each thread has its own registers and stack, which are important for its independent execution. The spawned threads therefore run in parallel with the main thread due to which no blocking occurs.

Results

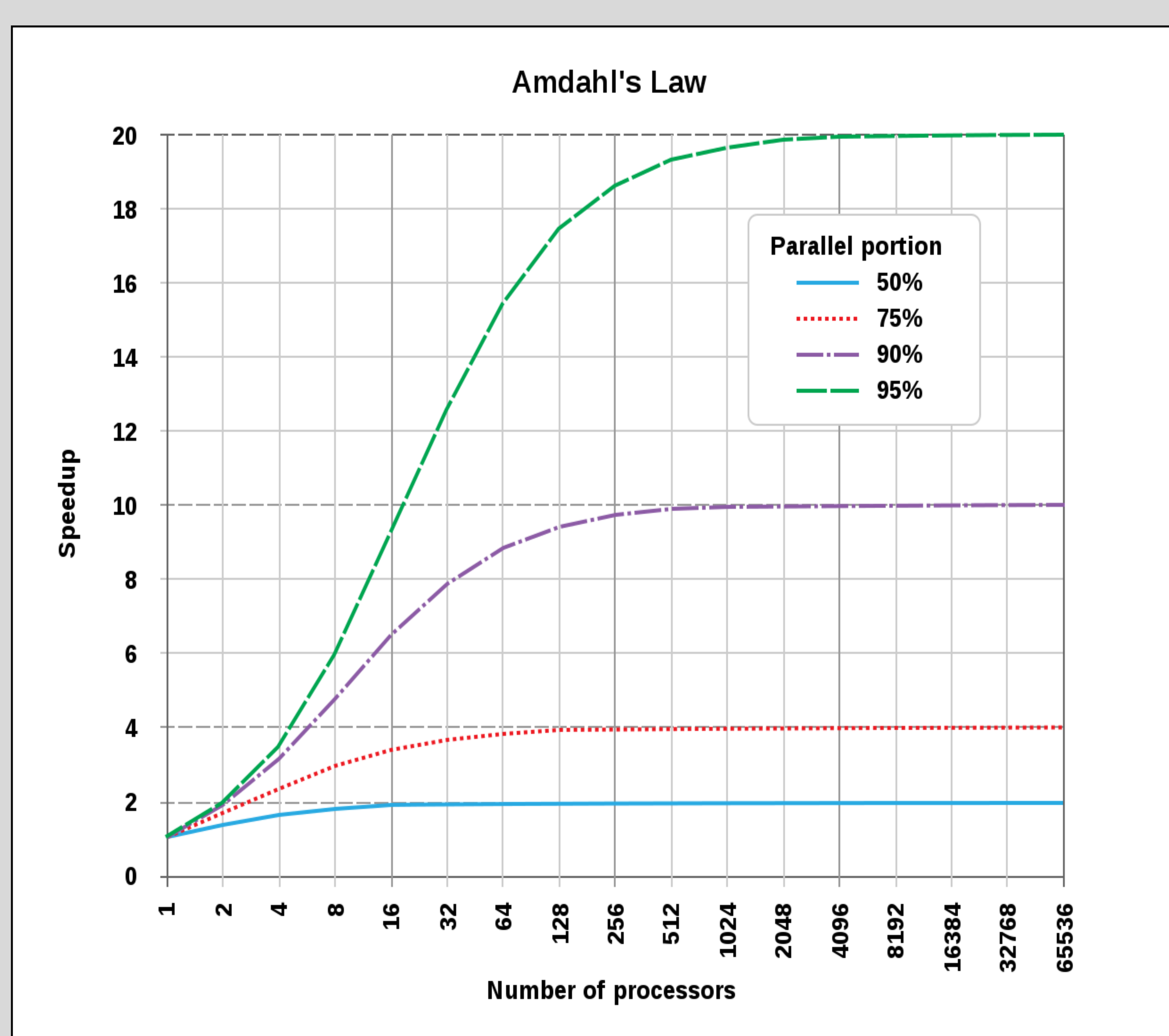

Figure 7: Amdahl's law [2]

Figure 7: The results achievable by multithreading can be estimated using Amdahl's law. According to this law, optimization efficiency is based on portions of parallelizable and sequential code and number of cores

Implementation

processKPImodule.ts

```
import { expose } from „threads/worker“
expose(processKPI);

async function processKPI(kpi:KPI):Promise<KPI>
{
    // KPI processing business logic
}
```

main.ts

```
import { spawn, Pool, Worker } from „threads“

main()
{
    var kpis = await db.get(/*get kpis with data for evaluation*/);
    var threads = Pool(() => spawn(new Worker("./processKPImodule“));
    var promises[];
    for (kpi of kpis)
    {
        let promise = threads.queue(w => w.processKPI(kpi));
        promises.push(promise);
    }

    var kpi_results[];
    for (promise of promises)
    {
        let KPI_output = await promise;
        kpi_results.add(KPI_output);
    }
    await db.save(kpi_results);
}
```

Figure 6: Node.js multithreading source code example

Figure 6: Multithreading can be achieved using *thread.js* library. The upper module describes the behavior of individual threads. The bottom module defines the behavior of the main thread.

sources of images:

[1] https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/4_Threads.html

[2] https://cs.wikipedia.org/wiki/Amdahlův_zákon#/media/Soubor:AmdahlsLaw.svg